2012.09

PROGRAMMER 程序员

运维自动化

ISSN 1672-3252
9 771672 325128 09>
邮发代号：2-665　定价：15元
www.programmer.com.cn　www.csdn.net

Contents

P22

资讯

封面报道：IT运维自动化

本期封面报道将从IT自动化的出现和发展谈起，讲述从小型创业公司到大型互联网企业的最佳实践，角度涵盖性能监控到软件部署，介绍最新的IT自动化运维工具和应用，为不同背景关注IT运维的读者提供参考。

管理

Contents


P131

移动


P134

云计算

2012年09月刊　总第239期

Contents

主管：中国社会科学院
主办：中国社会科学院文献信息中心
出版：《程序员》杂志社
网址：http://www.programmer.com.cn
国际刊号：ISSN 1672-3252
国内刊号：CN11-5038/G2
邮发代号：2-665
广告经营许可证号：京东工商广字0188号

总编：黄长著　Editor-in-chief：Huang Changzhu
社长/常务副总编：张悦校　President：Zhang Yuexiao
副社长：蒋 涛　Vice President：Jiang Tao
编委会：黄长著　张悦校　陈洋彬　蒋涛　曾登高　刘江
Editorial Member：Huang Changzhu　Zhang Yuexiao　Chen Yangbin
Jiang Tao　Zeng Denggao　Liu Jiang

执行总编：刘江　Executive Editor-in-chief：Liu Jiang
执行主编：孟迎霞　Managing Editor：Meng Yingxia
编辑部主任：董世晓　Director：Dong Shixiao
责任编辑：陈博　杨爽　卢鹄翔
Editors：Chen Bo　Yang Shuang　Lu Dongxiang
特邀编辑：蔡学镛　池建强　冯大辉　高博　肖新光　杨栋　余晟　周爱民
Contributing Editors：Cai Xueyong　Chi Jianqiang　Feng Dahui　Gao Bo
Xiao Xinguang　Yang Dong　Yu Sheng　Zhou Aimin
美术设计：纪明超　Art Designer：Ji Mingchao
美术编辑：朱凯　Art Editor：Zhu Kai
流程编辑：白羽中　Coordinator：Bai Yuzhong
Tel：010-64351458
E-mail：editor@csdn.net

广告总代理：北京创新乐知信息技术有限公司
Sole Advertising Agency：Bejing CSDN Co.,Ltd
Tel：010-64376055
E-mail：ad@csdn.net
Marketing Dept：010-51661202（ext 149）
E-mail：market@csdn.net

发行部
Distribution Dept.010-64351431
E-mail：reader@csdn.net

读者服务部
Readers service Dept.
网上订购：http://dingyue.programmer.com.cn/
读者信箱：reader@csdn.net
地址：北京市朝阳区广顺北大街33号院1号楼福码大厦B座12层
Address: B-12th Floor Fairmont Tower NO.33 Guangshun North
street,Chaoyang District,Beijing
邮政编码：100102
电话：010-64351436
传真：010-64348545

法律顾问：北京中润律师事务所 王杰
Law Consultant：Beijng Zhongrun Lawyer Firm
印刷：北京盛通印刷股份有限公司
Print: Beijing Shengtong Printing Co., Ltd.
出版日期：每月1日
Publication Date：the first day per month
零售价：RMB 15.00元　新台币 390元　HK $ 35.00（港、澳）
US $ 9.00（海外）
Retail Price：RMB 15, NT$390, HK $ 35.00,US $ 9.00

Wired

2012.8

乔布斯的故事：带来启迪还是发人深省？

乔布斯逝世已近一年，但他的传记仍然是最畅销的书籍。实际上，他的生活故事已变成企业家的圣经——是真理，也是异类。对于一部分人来说，乔布斯的生活揭示了无视员工或事业伙伴的心灵伤害而坚守个人远见和目标的重要性；对于另一部分人来说，乔布斯扮演着警世者的角色，一个改变世界的人却与周围所有的人格格不入，代价不可谓不大。这两种截然不同的反应证明我们的内心深处隐藏着两种往往相互矛盾而又推动我们前进的渴望：我们既想在工作中获得成功，又想让家庭和家人满意。对于那些像乔布斯一样誓言“活着就是为了改变世界”的人，乔布斯坎坷不平的生活故事迫使他们思考：做一个像乔布斯那样的人，真的值得吗？

甲方阵营姑且称为追随者。这是一群商人，在他们看来，只要过上乔布斯那样的生活，就拿到了成为更积极进取的远见家、竞争者的执照，归根结底，证明自己已成为积极进取的老板。他们放任自己陶醉于成为将军（有时是成为独裁者）的兴奋。工作早已是他们生活的中心，而乔布斯的故事让他们决心破釜沉舟。

乙方阵营姑且称为反对者。这是一群企业家，他们在乔布斯逝世后读了乔布斯事迹，自那以后就止步于乔布斯的整体构想——不仅止步于乔布斯对待员工的方法，而且止步于乔布斯对待生活那种毫不迁就的独裁方式。反对者都知道，克制住自己那些类似于乔布斯的倾向将会是一场苦斗，他们天生是奋斗者、完美主义者；他们还知道，撤出这场苦斗——接受以生活方式为中心的商业模式，意味着他们永远不会拥有走其他模式可以拥有的成就，更不用说拥有和乔布斯一样的成就。

乔布斯已成为罗夏墨迹测试，企业家和管理者可以将各自的追求投射其上。“每个人的内心中都有自己的乔布斯”，《好老板，坏老板》一书作者Sutton说道：“人们谈论的其实不是乔布斯，而是他们自己。”

Will Wright的创意：生活的游戏

差不多有30年了，Will Wright用他的创意游戏吸引着本来绝不会玩视频游戏的人。他还巧设游戏开发策略，让铁杆粉丝难以割舍，让游戏新手变成狂热玩家。秘诀在于，在Wright的世界里，没有输，也没有赢，有的只是游戏。

《模拟城市》、《模拟人生》等游戏让Wright名闻遐迩。这些游戏分别依托高深的建筑理论家、城市规划师、天体物理学家的不同作品进行制作，却毫无例外地让人瘾头十足。迄今为止，这些风靡一时的游戏已售出1.8亿套，为他人所不能，这要归功于他提出的“可能性空间”概念——在游戏中指玩家可以做出的行动或反应范围，大多数游戏的“可能性空间”还比较狭窄。

1987年，Wright成立了Maxis开发工作室，他与团队成员通力合作，摆脱种种约束，创造出一个无限灵活的游戏世界，玩家的技巧和想象力是唯一的限制因素。在Wright最好的作品中，玩家自由自在，自己决定自己的目标，游戏玩家和游戏设计者之间的分界线因而变得模糊不清。

2009年，Wright动手组建一家娱乐开发智库——Stupid Fun Club（傻乐俱乐部）。《连线》杂志与Wright一起回顾过去，展望未来，思考属于玩家和非玩家的前景。关于未来的游戏与生活，他们这样认为。

《连线》：你认为数据可视化将在我们的生活中扮演日益重要的角色吗？

Wright：既然我们能够获得如此之多的个人信息，我认为数据可视化是人们将会开始发展的一种文化知识。人们将对理解、解释这些信息兴致勃勃。这关系到人们如何想方设法洞悉自我，就像除了通过更多经验方式洞悉自我以外，人们还会通过占星或心理学测试洞悉自我。

《连线》：这两种概念看来都关系到互动，互动在我们的行为中正日益变得不

可或缺，未来的游戏会怎样影响我们的生活？

Wright：我们的身边时刻有人在讲故事——你会在饮水机边给你的朋友们讲故事，你可能在电视上看到一个故事，你可能去电影院看别人专业演出的好玩故事。游戏正在发展成我们生活中同样普及的事物。

卡巴斯基助克里姆林宫击败美国间谍

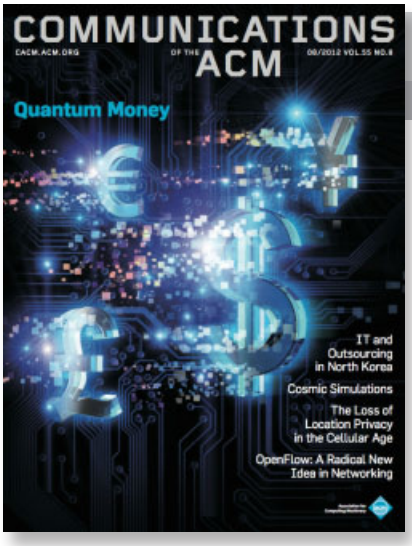
2012年6月1日，《纽约时报》首次透露白宫确实已下令部署一宗策划周密的网络间谍和阴谋破坏行动，目标是反德黑兰，Stuxnet是其中一个环节。6月19日，《华盛顿邮报》有把握地透露，Flame是这场反伊朗地下战的另一环节。但卡巴斯基实验室已截获了这个病毒——实际上是毁了它。

卡巴斯基实验室及其主人卡巴斯基的力量十分强大，据《福布斯》杂志称，在2009年到2010年间，卡巴斯基反病毒软件零售额增长率达177%，每年几乎新增用户450万，接近其竞争对手Symantec和McAfee之和。微软、思科和Juniper网络公司纷纷在产品中嵌入卡巴斯基代码，卡巴斯基公司因此卓有成效地收获了3亿用户。

从各个方面来看，卡巴斯基实验室与克里姆林宫的关系均等同于美国大型安全公司与华盛顿的关系。莫斯科向卡巴斯基支付数百万美元，令其保护政府网络安全，而五角大楼则向McAfee和Symantec慷慨撒金；卡巴斯基帮助FSB发现网络阴谋，McAfee和Symantec则与FBI携手合作；卡巴斯基的员工向俄罗斯议会发号施令，美国研究人员则向美国国会和白宫指手划脚。

卡巴斯基和莫斯科政府对网络安全的观点惊人地相似，已超出了安全行业的基本任务——保护数据安全的范围。当卡巴斯基和克里姆林宫官员谈起对在线威胁的反应时，不仅谈到对恶意数据加以限制，而且希望对他们眼里的恶意信息加以限制，包括那些可能引起骚乱的字眼和想法。

对于卡巴斯基来说，曝光Flame反映出卡巴斯基公司具有更大雄心：成为全球犯罪终结者兼和平捍卫者。卡巴斯基说，恶意软件已从小打小闹发展成犯罪工具、国家武器，因此，很自然的，他和他的反恶意软件战士的影响力也随之发展壮大。“我的目标不是赚钱，钱就像氧气一样，够用即可，拯救世界才是目标。”



Communications of the ACM

2012.8

手机时代：位置隐私荡然无存

蜂窝电话向来是一项监控技术。蜂窝网络的设计目的是追踪电话位置，从而将来电路由到最合适的基站（通常是最靠近用户的基站）。这项功能起源于美国手机紧急呼救服务E911。1996年，美国联邦通信委员会通过努力，加强了手机911呼叫位置解决方案。E911确定了一个要求，即在手机用户用手机拨打911时，手机服务供应商要向美国公共安全应答站发送位置信息。E911的意图很明显——如果紧急救援人员不必全面搜索手机站覆盖区域就能定位遇难者，那就理想了。可是，过去16年多以来，它对技术和社会的影响已远远超越它的原意。

E911更为直接的结果之一是，许多蜂窝手持设备都具备某种形式的GPS功能。服务供应商日益认识到，有了这项功能，就可以提供更丰富、更赚钱的位置服务。但应该理解，GPS的设计初衷并不是为了手机应用，而是为了户外应用。构成GPS系统的24个太空航天器发射的微弱信号很难在室内探测到，高大的建筑阻断了信号。GPS的设计目的还包括自动接收器的使用，GPS信号经过调制可以向接收设备提供太空航天器位置和轨

道信息。

2011年4月，苹果和Google公司通过WiFi和基站获取地理位置的功能引起轰动。iPhone会记录Mac地址和检测到的访问点的信号强度，然后加上时间标记和地理标记。通过从大量iPhone获得这样的数据，可以得出基站和访问点位置的高精度估计值。

手机位置估计值的精确度已达到了临界点：服务供应商利用访问点和基站位置信息，可以获得地址级精度的位置估计值。这些估计值的使用带来了严重的隐私问题，用户行为、喜好和信仰信息会严重泄露，进而威胁到用户安全和自主权。《Communications of the ACM》有文章使用Shannon理论单一距离概念对相应问题进行了讨论。

OpenFlow：激进的互联网新创意

从历史角度看，计算机网络的发展可谓各自为营——不同的网络元素各自占据着特定的网络生态领域，包括路由器、交换机、负载平衡器、网络地址转换及防火墙等。软件定义联网（Software-defined networking）这个概念意图颠覆了这种网络生态，从而将网络转化为一个整体平台，将不同的网络元素转化为各种可编程体。运行于这个网络平台上的应用可以对通信流进行优化，以便取得最短路径，这种功能与目前的分布式协议相一致，不过应用还能对网络进行优化，实现链接利用最大化，为不同用户创造出不同的可达域，或实现设备移动的无缝性。

OpenFlow，一种允许在IP网络中实现软件定义联网的开放式标准，作为一种新型网络技术，将带来许多网络新应用和网络管理新方式。OpenFlow的基本理念是：如果路由器是虚设的，并从一个大型“空中路由编译器”（Route Compiler in the Sky，简称RCITS）下载路由信息，则路由器所请求的CPU就会成为其端口速度和数量的函数。RCITS不必属于特定供应商，供应商则需要以最佳RCITS软件相互竞争，优胜劣汰。

很明显，事情并不那么简单。冗余之类的问题必须加以考虑，也就是需要有多个RCITS，还要有故障恢复机制。路由器需要具有足够的智能，要懂得如何在自己和RCITS之间进行数据路由。同样，实体之间的通信渠道也要保证安全。OpenFlow标准解决了这些问题。

OpenFlow的设计目的是在单一组织中使用。OpenFlow中的所有路由器作为一个整体进行工作，控制器对它控制的所有路由器具有无上的权力。OpenFlow不会取代因特网服务供应商内部系统，像一切新技术一样，它的转换不会突然发生，甚至有可能完全不被接受。OpenFlow同样有可能引起混乱，基于网络应用的民主化是一个疯狂的概念，会产生一些稀奇古怪的应用。可是有一天，当我们回想过去时，当时的一切会让我们嗟叹不已吧。

量子货币

有了现金和信用卡，你随时可以想买什么就买什么。那么为什么量子计算理论家要重新考虑钱的问题呢？

目前，货币有两种基本类型：一是实物货币，包括纸币、硬币、贵金属、地铁票等；二是委托他人保管的货币，如银行账户和信用卡等。但从理论上讲，任何实物货币都能造假，只要使用与原制造工艺相同的技术即可；而信用卡的缺点是，付钱时需要告诉银行把钱付给谁。这样一是会留下存根，二是一旦与银行断开连接就无法使用。这两种货币种类都不理想。人们真正需要的是既可以不留痕迹地花费，又能轻松大量携带的货币，这种货币应该是数字式的。

量子力学来了。物理学家多年前就知道，如果谁拥有一个量子物体，同时对它一无所知，那么它基本上无法被完全复制。这被称为不可克隆定理，这给我们带来了希望——量子信息可以成为更好的货币品种的基础。

我们将量子货币分为两大类。较简单的一类是，由造币厂生成一些由量子位组成的量子钞票，谁都可以保存、携带这种量子钞票，甚至还可以用它交易，换取他物。需要验证量子钞票的有效性时，可以向造币厂发送量子位，造币厂通过保密程序检验量子钞票是否处于正确状态。在这种架构下，除了造币厂，谁也不知道如何检验量子货币。这叫做私人密钥量子货币，因为用于检验货币的信息是属于造币厂私有的。

另一类量子货币叫做公共密钥量子货币。造币厂生成量子状态，任何人都可以携带或花销。任何人都可以自行检验货币，不用与造币厂通信。如果公共密钥量子货币能够实现，将成为前文讨论过的理想货币。

未来会有政府以量子货币代替现有货币吗？也许会。你可以使用量子货币在线购物——不用付手续费，也不会受到第三方监管；你可以将量子货币下载到qPhone（商标尚未注册）并从量子贩卖机上买东西。随着量子货币的降临，诚祝大家更享受花钱的乐趣。

（感谢译者李芳支持）

捷径的教训

文 / Rob Pike 链接：bit.ly/ReWqhC

本文作者Rob Pike通过Unix文件系统中一个古老的小问题告诉大家：贪图一时便利可致遗患无穷。

很久以前，Unix文件系统的设计逐渐成形时，为方便不同目录间的切换，“.”和“..”这两个文件入口应运而生。可这样一来，输入“ls”命令时这两个项目也会列出。于是，不知道是Ken还是Dennis在程序中加了个简单的判断，那时是用汇编写的，但我们要讨论的代码与如下的C代码等价：

```
if (name[0] == '.') continue;
```

代码本来应该这样写：

```
if (strcmp(name, ".") == 0
    || strcmp(name, "..") == 0)
    continue;
```

前面的代码不过简短一点，后面的其实也很简单。于是两件事发生了：

首先，前面的代码树立了坏榜样。许多懒惰的程序员仿效之，在代码中做同样的简化，结果引入了软件Bug，名字以点号开头的文件常在不该忽略时也被忽略了。

然后，更糟糕的是，这一做法带来了“隐藏文件”这一创意。结果，更懒的程序员开始在所有用户的home目录中放置文件。就拿我的电脑来说，安装的软件并不多，可我的home目录中却有上百个点号开头的隐藏文件，其中大多数我都不知道是做什么的，也不知道是否还有用。每次遍历home目录做文件名运算时，操作都变得很慢，就因为这一堆烂泥。

我可以很有把握地说，“隐藏文件”的概念是个意料之外的结果，显然是个错误。

四十年前抄捷径，问题知多少。它究竟引发了多少软件Bug？浪费了多少处理器时钟周期？又给人们带来了多少沮丧？更不要说导致的不良设计了。

下次当你准备在代码中抄近路时，请想想这个教训。

Valve管理模型的政治经济学分析

文 / Yanis Varoufakis 链接：bit.ly/O5YVSy

继《Valve新员工生存手册》和Michael Abrash关于在Valve工作的精彩记述之后，本文作者Yanis Varoufakis通过独特的政治经济学视角论述了Valve的管理模型，深入分析阐明了Valve的内部组织结构及其运作机制，更引发了对公司制度未来发展的思考。

公司：市场中的自由区域

经济活动中有一座堡垒，一直在顽强地抵抗着市场的胜利，那就是各种公司。想象一下：公司几乎是市场社会的代名词，可与之矛盾的是，公司本身却是“市场中的自由区域”。公司内部也有稀缺资源的分配，然而却不通过价格手段，而往往通过一种等级式的机制。

变革之轮：Valve新式“自发秩序”的终极符号

如果有人问我Valve的符号是什么，我会回答一只滚轮，就是Valve每台办公桌都配备的那种。它使我们可以在公司内自由移动，加入我们想加入的工作组，或者自发形成新的组，而无须任何人批准。至少在我眼中，这一滚轮体现了Valve在公司内部创建“自发秩序”的尝试，这种秩序并非基于价格信号，而是基于Valve的员工通过移动滚轮而发送给彼此的一种分散式、个体化的时间分配信号。

基于时间分配和团队形成的自发秩序：Valve方法

Valve启用了时间和团队的分配信号，以此形成自发秩序。每位员工自由选择其团队伙伴，以及在各个参与竞争的项目上投入的时间。只需从一般员工身上，以及兴趣、能力与自己接近的员工身上观察不同项目和团队的受欢迎度，员工便可取得关于项目和团队的大量有用信息。

通过这样一种不断进化的过程，员工的能力、潜质和思想都得到了最佳的发展机会，从而产生了一种提升共同利益的协同作用。就像有一只看不见的手在指导着Valve的每位员工，使他们做出既能发挥个人潜力又能服务于公司利益的决策。

结论：Valve传给未来的信号

极端浪费人类才能和精力的等级结构，再纠缠上毒药一般的金融，勾结上正迅速失去民主合法性的政治结构，现有资本式的公司制度正在走向消亡。一种后资本主义的，非集中式的公司形式迟早会出现。

销售员和开发者

文 / Daniel Tenner　链接：bit.ly/R0MaW0

本文作者Daniel Tenner通过一个猎熊的寓言巧妙地阐释了销售员和开发者的关系，以及他们对企业的价值。

一位销售员和一名开发者一起去猎熊。他们到达了森林小屋并开始卸车。销售员很快厌倦了搬东西，对开发者说："你继续卸车，把一切准备好，我去找只熊来。"开发者习惯性地叹了口气，点了点头。

半小时后，开发者差不多收拾完四分之三，当他走出小屋时听到了猃猃的叫声。二十米外，灌木摇动，销售员从中飞出，身后一只怒气冲天的巨熊淌着口水。

开发者赶紧躲到屋外一把椅子后面。销售员直冲向小屋，巨熊紧随其后，快到门口时销售倏地闪向一边，熊却一下子撞进门里，销售员敏捷地关上了门。熊在屋里扑腾的巨响外面听得一清二楚。

开发者从椅子后面走过来，销售员欢呼着说："这是第一只。现在你去宰了它，把皮剥下来。我再去找一只！"

两种视角

解读这个故事可以有两种方式，倾向哪一种在很大程度上取决于你是构建者类型还是销售类型。

如果你是构建者类型的，你会觉得这个故事恰如其分地例证了与销售打交道过程中的典型问题：他们签下单子便不再关心后续进展。实际上交付一个项目并非易事，但那时销售已转去做别的事情，因此他们不关心。然而，如果你是销售类型的，便会有不同观点。这不过是又一则取笑销售却完全不考虑实际情况的故事，找一只熊，把它带回来再引进门哪有那么容易。

销售并非可有可无

如今许多创业者都是从开发者转型而来，他们倾向于将销售看成次要的事情，认为等做到那一层再考虑也不为晚。

然而，销售可不是次要的。我也是构建者类型的，但经历过一些业务以后，我现在相信有人去寻找愿意掏钱的客户是非常重要的事情。如果找不到这样一个人，我宁愿不创立这家公司。

没有人积极推销一款产品、一种服务或任何你想售出的东西，销售便不可能发生。有人梦想好的产品可以推销自身，但即使像Dropbox和Apple II那么棒的产品也需要严肃对待销售工作方能打开销路。

是非成败转头空：我与SWG

文 / Karl Parakenings　链接：bit.ly/RfeN63

本文作者Karl Parakenings记述了自己与星球大战游戏（Star Wars Galaxies，以下简称SWG）之间的恩怨：一款曾经深爱的游戏，最终却帮忙毁掉了它。

2001年夏，我开始钻研这款即将到来的游戏。它听上去棒极了。离公测还很遥远，但已成形的在线社区让我产生了浓厚的兴趣。

2002年春，第一版沙盒阿尔法构建开始测试。有一天，我去咨询游戏的经济结构将是怎样的。得到的回答改变了我的生活，他们说："实际上我们并未做太多规划，玩家会把它组织好的。"

经过周密准备，发布那天我一早跑去买了8份拷贝，然后一溜烟回到家。我请了一周假，空卧室里储满了食物和饮料，一排电脑排列成了半月形……

通过虚拟货币赚到实际货币。这太神奇了，而一切都运作得非常好。我清楚地记得，有一天我意识到自己从这上面挣到的钱已超出了我的全职工作收入。我很快决定扩充规模，雇了四个小伙子在新加坡24×7地玩游戏。很快钱赚得更多，我需要再一次扩充，最多时我雇过12个人。

我太太也渐渐转变了态度，一开始她讨厌我对游戏的沉迷，慢慢地她开始帮我校对数字，还提供其他建议。作为回报，我为她买了一辆轿车，还买了一栋房子。

大约两年以后，我发现这无法持续了。玩家数量在下降，游戏的管理越来越乱。当他们去掉了holo-grinding，我的商业模式便被极大破坏了……

游戏在2005年的NGE/CU之后就死掉了。当开发者们背弃了曾付出努力的玩家，转而去顺从试图直接索取一切的懒虫所发出的抱怨，游戏就已死了。讽刺的是，正是这些抱怨的人曾经很欢喜地花钱买我的点数和Jedi账号。

我是否促成了SWG之死？是的，我很早就接受了这一事实。一款曾经是我深爱的游戏，我帮忙毁掉了它。

（感谢译者王江平支持）

TextMate 2开源

TextMate是Mac平台下的著名文本编辑器，最初由Allan Odgaard于2004年发布，尽管当时只包含非常有限的功能，甚至连“偏好设置”和“工具栏”都没来得及实现，但因为拥有Bundle等创新特性，并且与操作系统内置功能良好融合，还是受到了一批开发者拥护。在此之前，BBEdit已统治Mac平台编辑器领域十余年，却鲜有让人眼前一亮的新功能。TextMate的发布，为这个沉寂的领域带来了一缕清风。

随后TextMate进入了快速开发期，2006年1月1.5稳定版发布后，连之前一直批评它的人也转变了态度。是年8月，它获得了对Mac应用软件的最高褒奖“Apple Design Award”。

盛极一时之后，TextMate却陷入了BBEdit曾经的状态，Allan许诺的项目管理、远程文件编辑、版本控制功能进展缓慢，接下来的整个2007年软件核心几乎没有变化，虽然此时它的Bundle功能仍然领先。这一年，Allan在被问到是否会将TextMate开源时，他给出了否定的答案：“我希望软件核心部分按照自己的意愿开发，而且我担心开源赚不到钱”。也是在这年，Jon Skinner辞去了Google的工作，开始专心开发Sublime Text。

2012年伊始，当Sublime Text 2、Chocolat这些编辑器新秀开始发力时，TextMate才终于勉强拿出了2.0测试版，并在8月10日出人意料地选择了开源。不论这是否如Allan所说，选择开源是深思熟虑的结果，但6年间，它的版本号始终定格在1.5。

与移动和互联网领域应用激烈的竞争相比，桌面软件似乎平静许多，然而当你的进取心不足之际，或许在某个角落，竞争对手已在“鸭子划水，暗中使劲”。

别再谈论盈利模式了

移动新闻应用《The Daily》裁员三成，Onlive面临破产危机，“移动互联网盈利难”这朵乌云愈发让开发者们喘不过气来。有趣的是，我们在参加移动互联网相关的会议、论坛分享及各类采访中，时常会看到创业者讨论有关“盈利模式”的问题。更有趣的是，一群人讨论来讨论去，移动互联网的盈利模式还是那么几种：付费下载、应用内付费、广告、与运营商/终端厂商的合作分成等。

随着巨头进驻，通用领域市场机会已非常渺茫，以量带动的广告模式越来越不靠谱。相信谁都不太愿意为自己完全不了解的产品一次性付费，而一般的创业者也很难得到与运营商、终端厂商合作的机会。目前看来，应用内付费模式是当前环境下的最佳选择。

因此，真正重要的问题不在“盈利模式”上，而是如何在你选择的“盈利模式”下去制定“用户模式”和“产品模式”。“用户模式”即选择你的目标用户，他们的年龄、特点、喜好、习惯，它决定了用户会为什么样的产品埋单。而“产品模式”显然要与“用户模式”匹配，包括了整个产品的设计、宣传渠道、定价等。例如在某些游戏产品中，以群聚型玩家为主的游戏加强群体活动的比重，而以高付费意愿玩家为目标的游戏力图筛选用户打造“精英圈子”，它们的用户群不同，却都能获得不错的回报。

当然，不是所有产品能都完美匹配当初在“用户模式”和“产品模式”上的设想，如果在实践中发现你的产品定位发生了改变，这时再来探讨是否要重新选择“盈利模式”也是有必要的。

“基于呈交给苹果诉三星案的早期证词和法庭文件——包括照片、电子邮件和原型机——我们开始明白乔布斯到底是如何施展他的魔术了。”

——《纽约时报》记者Nick Bilton在谈到苹果同三星的专利之争时说，苹果用陪审团庭审的方式捍卫自己最有价值产品的专利，同样也使得竞争对手和公众有机会一窥这家世界上最神秘公司的究竟。

“从2009年开始，37signals Job Board上的远程办公职位的比例变得越来越高。”

——37signals创始人之一Jason Fried在Twitter上谈到了他的新发现。Jason一直认为远程办公对创业公司来说利大于弊，在他做过的一个TED演讲中曾说“办公室根本就不是工作的场所”。

“Metro只是产品开发过程中的一个代号。因为临近产品正式发布，以及需要将话题从面向专业人士转到消费者，我们将使用自己的商业名称。”

——在谈到微软为何突然宣布将放弃Metro这个名称时，发言人做了这番表述。这个名词已经完成了历史使命，现在，该把宣传的重点放到真正的产品上了。

“我相信纯文本故事本身就是一种媒体形式，而它将延续下去。打个比方，我不认为加点音频片段会让海明威变得更好。”

——亚马逊CEO贝索斯认为，设备一旦定位为多用途，就无法在单一用途上做到极致。因此他相信，尽管平板设备会越变越好，但专用阅读设备将与其长期并存。

“Outlook是私人的，资料完全由你掌控，你的私人通信不会被拿来进行广告用途。”

——目标性广告是网络上价值数十亿美元的生意，但微软在Outlook.com中放弃了对邮件内容的扫描，把加强隐私作为一个卖点，吸引不想被Gmail窥探的用户。

“就我们目前所了解的情况来看，Quest将是Dell未来软件架构的基础，技术上依然会保持中立发展线路。”

——硬件起家的Dell将试水软件领域，以24亿美元的高价对Quest Software完成收购。在谈及这次收购时，Dell高级副总裁Steve Dickson如是说。

“考虑到Tumblr与Twitter合作的历史，我们对Twitter的这项决定真的非常失望。”

——Twitter宣布对一些应用施加新的限制，或将是其诞生以来最为大胆也最具争议的行为之一。这个第三方应用数最多的开放平台已全面转向封闭。

CSDN十大资讯

2012年7月20日–8月20日

01 Android正式禁用Flash

作者 / T.C. Sottek

Flash与HTML5之争由来已久，但最近两年，这场战争的形势日趋明朗。8月15日，Adobe宣布所有Android设备正式禁用Flash，这也意味着，Flash在HTML5面前低头屈服。Flash棺材上的最后一根钉子，很可能来自微软。此前，微软表示计划在Windows 8中对Flash提供有限制的支持。

02 《Warcraft》是怎样炼成的

作者 / Patrick Wyatt

《Warcraft》究竟怎样从一个想法进化为一个完整的游戏？它的创造者Partrick Wyatt表示，概念的实现永远都不是一条直线。与其他游戏一样，《Warcraft》的设计过程中，不同成员间一直有很多争论，不满和道歉贯穿着始终，但这并不总是坏事。

03 月薪2000美元，多了还是少了

作者 / 夏梦竹

Hacker News的一篇文章，提出了这样的问题：月薪2000美元，100%的自由时间，地点也由你来定，这样的工作环境你想要吗？从讨论的情况看，时间完全可由自己支配，似乎对程序员们并没有多大的吸引力。而能否过上安逸的生活，则很大程度上取决于当地的物价水平。

04 后Hadoop时代的Google三驾马车

作者 / Cade Metzemail

Google在2003年到2004年公布了关于GFS、MapReduce和BigTable三篇技术论文，成为后来云计算发展的重要基石。如今，Google的Caffeine、Pregel、Dremel或将再次影响着全球大数据技术的发展潮流。如果你想了解未来技术趋势如何，请别错过关于它们的论文。

05 最有价值的编程忠告

作者 / Rob Pike

在与Ken Thompson共事的过程中，Pike认识到“纠错前先思考”。如果你一头扎进问题中，可能只解决了当前代码的问题，但如果你先思考这个错误、这个Bug是怎样引入的，通常你会发现并有可能纠正一个更高层次的问题，从而改进了系统设计。

06 是时候取消软件专利制度了

作者 / 魏兵

专利之争一直在各大科技公司之间不断地进行。软件专利的存在本身没有错，关键是握有专利的一方将用专利来做什么。Google 公共策略总监 Pablo Chavez 在接受采访时表示：“我们正在认真考虑一件事，现存的专利系统是否真的能够激励创新？”

07 国外作者笔下的中国创业指南

作者 / Jon Russell

TheNextWeb刊登了一篇中国创业指南的文章，从关键企业、投资商、孵化器、应用市场这4个方面剖析了中国移动市场的创业机遇和环境，文章结尾还提出一些他们认为值得信赖的关注中国市场的投资商。当局者迷，让我们看看在外国人眼中的中国创业图景。

08 危险了，世界赖以运行的软件

作者 / James Kwak

Bats因交易系统出现问题取消IPO；Facebook首次公开募股，纳斯达克无法确认订单长达数小时，瑞银因此而损失超过3.5亿美元；Knight Capital因系统出错也损失了44亿美元。那些本应支撑证券市场的关键软件，却成了影响世界经济命脉的不稳定因素。

09 Twitter颠覆者App.net首次募资成功

作者 / 魏兵

由Dalton Caldwell发起的移动应用发现平台App.net首次筹款目标已经实现。Caldwell的想法是网站不出现任何广告，只面向应用开发者收费，普通用户则可以免费使用，并利用网站找到他们想要的应用程序。但App.net能够否成为Twitter的颠覆者目前还很难说。

10 新标准在即，未来将用上1Tb以太网

作者 / Stephen Shankland

更快的以太网标准对普通百姓仍然重要。要知道，互联网连接的另一端是诸如Facebook、Google、电信和金融服务公司，它们对网络容量需求正快速增长。如果以太网的发展跟不上他们的需求，互联网速度要么变得越来越慢，要么费用就变得昂贵。

新编程行话

当你通过搜索引擎寻找问题的答案时，有多少次链接将你引向Stack Overflow，并让你最终满意而归？开发者在Stack Overflow交流的过程中，也创造了这个社区独有的氛围和行话，Jeff Atwood对这些行话进行了总结。

01 尤达条件语句（Yoda Conditions）

按照尤达大师那种“蓝色是天空的”表达习惯，if语句也要写成常量在前变量在后，例如if(4 == foo)。

02 埃及括号（Egyptian Brackets）

前半个括号放在一行结尾，也就是K&R风格的括号。想知道这和“埃及”有什么联系？请想想埃及壁画中人物的手势。

03 自以为是的报告（Smug Report）

常有用户高估了自己对系统的理解，Bug报告里充满与技术无关的细节和建议（而且总是错的），他还会指挥你，试图教你如何修复这些Bug。

04 反向重构（Refuctoring）

这一过程是将一段原本结构清晰的代码通过一系列细微的改变，最终使它变得除你本人之外，旁人完全无法维护。

05 海森堡Bug（Heisenbug）

海森堡的名字一直都和他的量子力学理论联系在一起，他提出了著名的“不确定性原理”。而海森堡Bug说的是那种一旦试图开始研究，就会突然消失或改变的Bug。

06 恐惧驱动开发（Fear Driven Development）

软件开发的主要动力来自于项目经理增加额外的压力（开除某人、提前截止、抽减项目资源等）。

07 九头蛇代码（Hydra Code）

代码无论如何也改不好，像传说中的九头蛇一样，每修补一处，就会引入两个新的Bug。这种代码应该被重写。

08 吉米（Jimmy）

这是对新手菜鸟的泛称。如果我们正在开发一个不需要多少背景知识就能为其他开发者所用的框架时会这么说：“如果吉米忘记更新属性怎么办？”

09 水怪Bug（Loch Ness Monster Bug）

像尼斯湖水怪一样，这种Bug不可再现，或者只被一个人发现过。但办公室的人都在谈论它，每个人都很重视。

10 叠叠乐代码

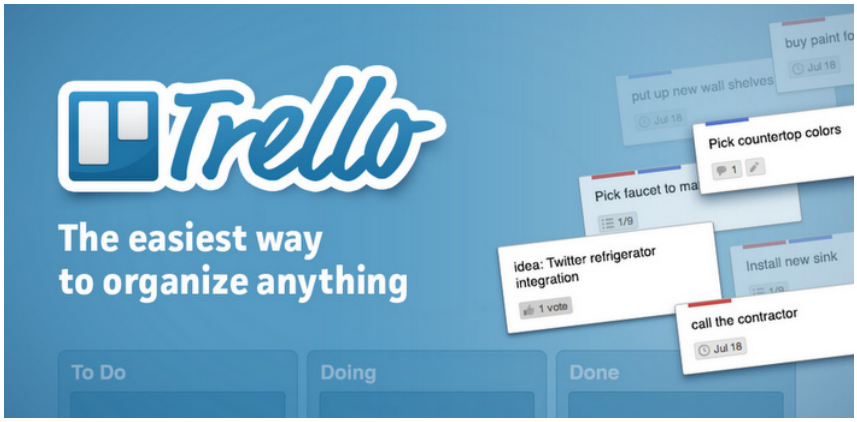
近似叠叠乐积木玩具倒塌前的状态——只要改变一小段代码，整个程序都会崩溃。

Trello Android版发布

Trello是一款团队项目管理Web应用，由Joel Spolsky的Fog Creek Software于2011年9月推出，目前用户数量已超过50万。

Trello使用了MongoDB、Node.js、Backbone.js等大量现代前后端技术。在支持移动设备方面，除了能够在浏览器中运行，此前已推出了iPhone客户端提供更流畅的体验。

利用新推出的Trello Android应用，用户除了可以完成创建卡片、留言、投票等Web端能够进行的操作外，还可以离线和浏览和接收推送消息。



Zsh 5.0发布

Zsh是一种Unix Shell，最初由Paul Falstad在1990年写成，当时他是普林斯顿大学的学生。据说Zsh得名于当时在耶鲁大学任助教的劭中，他的登录名正是“zsh”。

Zsh 5.0的变化主要包括：支持多字节字符、命令行高亮、非交互Shell及子Shell作业控制，Zshroadmap手册提供更有用的介绍，显著增强了POSIX兼容性，新增COMBININGCHARS、DEBUGBEFORECMD、HASHEXECUTABLES_ONLY等选项。

OpenGL ES 3.0

OpenGL背后的非盈利组织Khronos Group宣布，发布移动设备专用的OpenGL ES 3.0标准。OpenGL ES 3.0向后兼容ES 2.0，它让开发人员在支持硬件规格上更加容易，同时该标准也为移动设备带来了更多桌面版OpenGL 3.3和4.x标准的功能。其中最重要的就是新的纹理算法带来了更好的纹理压缩表现。

OpenGL ES 3.0的主要新功能包括：多重增强渲染管线，实现先进的视觉效果加速，包括遮挡查询（Occlusion Query）、变换反馈（Transform Feedback）、实例渲染（Instanced Rendering）；高质量ETC2/EAC纹理压缩格式成为一项标准功能，不同平台上不再需要不同的纹理集；新的GLSL ES 3.0着色语言，全面支持整数和32位浮点操作；纹理功能大幅增强，支持浮点纹理、3D纹理、深度纹理、顶点纹理等。

XMind 2012发布

XMind是一款国产思维导图软件，其开源版本曾经获得Sourceforge社区2009年“最佳学术应用奖”。它采用Java语言开发，可以在Windowns、Linux和Mac平台上运行；基于Eclipse RCP架构，可支持Eclipse形式的插件，通过编写XML清单文件可以扩展系统定义好的扩展点。

XMind 2012带来了重新设计的甘特图、全新版本管理，以及拼写检查视图等新功能。并对已有的功能、设计进行了几十项改进，修正了之前版本存在的漏洞。

Unity 4

Unity是一种用于3D动画和图形的流行游戏开发工具，在荷兰阿姆斯特丹举行的Unite 2012大会上，Unity公司CTO Joachim Ante介绍了新的Unity 4开发平台。

Unity 4开发平台包括了全新的Unity引擎和配套开发工具。Unity 4最受关注的新特性是收购Mecanim后推出的新动画系统。其他新特性包括优化粒子效率提升、增加GPU Profiler、减少移动平台Mesh内存消耗、支持动态字体渲染、支持移动平台的动态阴影等。

GDB 7.5发布，支持Go语言

GDB是GNU软件系统中的标准侦错器，最初由Richard Stallman于1988年开发。它能够运行在绝大多数的Linux、Unix和Windows系统中，可以对十数种处理器架构的应用程序进行调试。

GDB 7.5最重要的新特性是带来了对Go语言的支持。其他新特性还包括：增加了对x32 ABI、microMIPS、Renesas RL78等处理器架构的支持，改善对Python脚本的支持，增强的GDBserver，支持通过stdio等方式连接，以及在ARM平台上支持逆向除错诊断。

Bitey

为了提升程序性能，Python程序员经常会使用ctypes调用由C语言编译出来的函数，好处是接口简洁，兼容绝大部分的C类型数据结构。Bitey则更进一步，它能将LLVM的bitcode直接导入Python，并作为扩展模块使用。Bitey复用了ctypes的基础接口兼容大多数C数据类型，包括整数、浮点数、指针、数组和结构体。

下面是一段计算Fibonacci序列的C代码：

```
/* fib.c */
int fib(int n) {
    if (n < 3) {
        return 1;
    } else {
        return fib(n-1) + fib(n-2);
    }
}
```

使用clang将其编译为LLVM bitcode后，利用Bitey，不使用C编译器、连接器或动态装载器，也不需要写包装函数，便可直接将目标文件导入Python并运行。

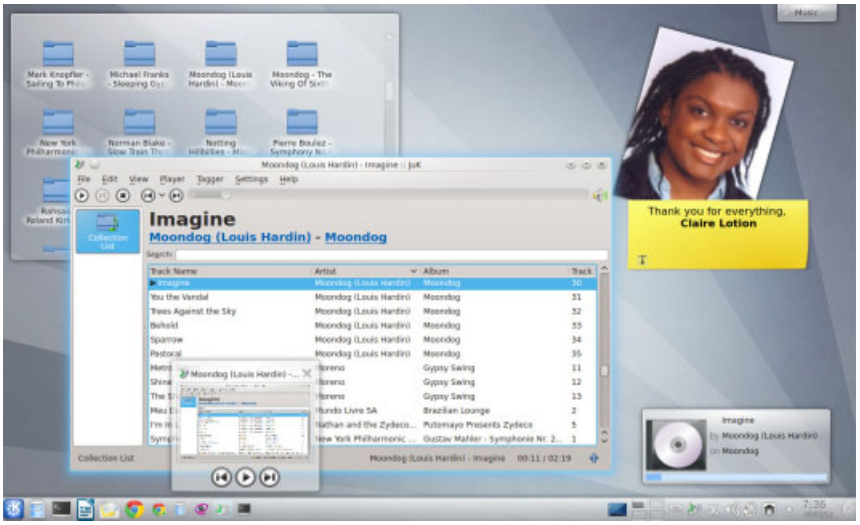
```
bash % clang -emit-llvm -c fib.c

>>> import bitey
>>> import fib
>>> fib.fib(38)
39088169
```

Twitter Bootstrap 2.1发布

Bootstrap是Twitter推出的一个开源前端开发的工具包，它由Mark Otto和Jacob Thornton合作开发。Bootstrap提供了优雅的HTML和CSS规范，它由动态CSS语言LESS写成，与CSS框架Blueprint存在很多相似之处。Bootstrap推出后颇受欢迎，一直是GitHub上的热门开源项目。

Bootstrap 2.1的新功能包括：下拉菜单支持子菜单；新的Affix JavaScript插件；新增表格行的状态类，以便更好地表达表格中的成功、警告和错误消息；为导航和下拉菜单添加了禁用状态；流式布局网格现在支持偏移，且系统变量不再是固定的百分比。



KDE 4.9发布

KDE团队将4.9版献给刚去世的KDE贡献者Claire Lotion，这一版对KDE Plasma工作区、KDE应用程序和平台都进行了更新。

Dolphin文件管理器能够显示更多的元数据，添加了新的Mercurial插件；Konsole支持通过KDE Web Shortcuts搜索选择的文本，提供“更改目录到”右键菜单，可以分离标签页；KWin能够在窗口切换时提升窗口，改善了Wobbly Windows的性能。Okular能存储和打印PDF文档中的批注，并可播放PDF文件中的嵌入视频。

Big Data标准的桂冠会落到 Hadoop头上吗？

如果一项技术不能及时、全面遵循业内标准，或者本身不是技术标准，那么它的影响力也会逐步消退，Hadoop也会面临这样的问题。

Hadoop似乎是全能的，因为它不仅被Google、Yahoo!这样的互联网巨头采用，就连JP Morgan Chase这样的“传统”企业也开始应用。Hadoop的繁荣似乎给人这样一个暗示——不久的将来，就连普通的便利店系统、进销存系统也会因为各类Big Data的存在逐步采用Hadoop。

但这十几年的技术市场也揭示了另一个趋势——即便一些技术产品普及广如Windows操作系统，如果它不能及时、全面遵循业内的技术标准，或者本身不是技术标准，那么它的影响力也会逐步消退。Big Data市场快速发展之下，也面临类似的问题，这不是Big Data产品提供商一方的事情，而是关系到能否形成一个围绕Big Data的良性技术市场生态圈的问题，因为这个生态圈还必须考虑到下游应用软件提供商、上游硬件提供商、技术教育和培训商等一系列参与者。没有标准可循将意味着Big Data产品提供商的产品可以更加嬗变，可以以自己个体发展需要，为了一部分上下游厂商忽略掉另一批厂商，使得一些新特性成为一批厂商契机的同时成为另一些关联企业的噩梦。

在众多Big Data“诸侯”中，Hadoop似乎成为最受欢迎的一个，不仅被微软、IBM和Oracle认可并集成到新的产品中，同时它也获得了前述一大批互联网巨擘的支持，以Hadoop为蓝本规划Big Data的标准似乎水到渠成，但是……

首先，标准往往在兼顾普遍性的同时，似乎更强调“先动优势”，即便“先来者”可能更加学术化、支持的产品和厂商没有市场上的“明星”备选者丰富，也不影响它成为标准。而Hadoop与他的同台竞技者们似乎都挺“谦让”。

其次，Big Data市场似乎还很年轻，大部分企业应用还是恪守“Don't buy into the hype（别被这些胡吹的东西骗了）”的原则。即便在美国，根据Forrester Research分析师James Kobielus的研究，只有不1%的机会将Hadoop用于企业生产环境，即便未来1年这个数字将会以2~3倍的速度递增，它仍只是一个“Up-and-Coming”的技术。

再次，Hadoop似乎还没有和标准中最关键的技术要求（例如哪些数据是Big Data，是否需要包括矢量数据，Big Data数据如何表示，Big Data采用何种方式访问等一系列问题）与同行达成一致，因为这些问题事关每个产品的核心技术架构，很可能决定一部分产品的生死。

然后就是到底什么是Hadoop“典型应用”的问题，各个大型Hadoop应用间的区别就好比白天与黑夜一样，我们看到的是各种各样的天才构思，但这些构思之间似乎都太“个性”，尤其在结合结构化数据对Hadoop中大量非结构化数据进行分析和发掘方面，这些解决思路简直就是“火花四溅”。这对于标准制定带来一些困难——到底怎样的分析方式才是应该相对固定下来，变成大家共同遵循的内容呢?

当然，Hadoop面临的这些问题其他竞争产品也在面对，作为当前市场最成功的产品，如果想成为Big Data标准的蓝本产品，最大的风险来自于时间，也就是那些也许正在加利福尼亚、班加罗尔、都柏林、深圳或珠海某个“作坊”里开发的产品，这些“英雄不问出处”的产品一旦面市并被软件巨头们相中，Hadoop的标准之路就会变得坎坷。为了能够“赶时间”成为标准并依托标准长期占据技术优势，Hadoop需要放宽胸怀与其他竞争者协商并包容各家的主要诉求。因为这不仅是它自身的事情，也是Big Data“场外”和“场内”用户的希望。

除此之外，Hadoop的标准之路可能还有一个关键抉择：以怎样的方式成为标准。对于绝大多数用户而言，Big Data只是一个补充，他们需要采用多种技术完成各类商业项目处理。而Hadoop在明确哪些是应该共同支持的Big Data类型之后，是应该“自立山头”还是被现有的数据技术标准“招安”，这就成了软件巨头间的博弈。从最终用户角度看，标准加入方式与使用方式一致也许是不错的选择，但在厂商看来，这个是明显的

利益之争。

为了自身和Big Data市场发展考虑，Big Data市场最有力的竞争者Hadoop应该承担起标准蓝本的角色，但它需要通过众多应用尽快成熟起来，想清楚、说明白、立规则。

王翔
软件架构师，主要研究方向为XML、.NET、领域设计和PKI应用。工作之余喜爱旅游、写作和烹饪。

发散的安全焦点

从议题的分布情况来看，国内的安全议题多半在传统领域，如Web安全、网银安全等；国外的同行们则带来了更多新领域的话题。

国内非常有特色的安全焦点系列会议xKungFoo和XCon八月在北京召开。安全界普遍认为xKungFoo是开场锣鼓，而XCon才是正会。以往XCon的议题基本云集于漏洞挖掘领域（而多数参会者也是为了在这方面取经而来），本届议题则非常发散，尽管攻击和防御依然是主旋律，但也多了取证、安全测试等方面的议题。

如果从议题的分布来看，国内的议题多半在传统领域，如Web安全、网银安全等；国外的同行们则带来了更多新领域的话题，如Paul Craig的《攻击图形化的受限操作环境》（并在会议上发布了新的工具）、Stefan Esser的《iOS内核堆利用战场》等。

微软的褚诚云带来了《Windows 8上安全对抗技术的提升》，他介绍了在Win8上代码编译、ASLR、堆和内核保护等方面新的安全措施。微软在安全方面的持续投入和长期战略，看起来正在不断结出果实。

iOS也注定会是一个热点，除了Stefan Esser，Andrey Belenko还带来了《iOS数据保护的进化和iPhone取证》的议题，这个议题覆盖了从iOS早期版本到iOS 5的数据加密策略的整个变迁。

安天实验室两位从事硬件安全研究的博士，在会上带来一个通过长波信号干扰原理改变电波钟和电波表的时间的演示。这个演示的原理并不复杂，但确实是过去的研究者罕有涉猎的一个点。我给这个报告打了8分，扣两分是因为演示之前的背景介绍环节过于冗长。

国内老牌研究者王伟的《负责任的安全漏洞披露》议题引发了同行的争鸣。关于漏洞的“No more free bugs”到底是不是正途的问题，预计还将讨论下去。但国内的研究者从经验、到方法论、再到产业责任的思考，却让我们看到一种宏观和成熟。

Jeffery Moss进行了本次的压轴演讲。他是ICANN副总裁兼首席安全官，BlackHat（全球技术性安全会议）和DEFCon（黑客大会）的创始人，美国国土安全咨询委员会顾问。而他报告的题目也具有一种辩证感“Why I don't trust anything（为什么我什么都不信）”。他列举了从ARPANET到移动网络、从软件到硬件、从互联网基础设施到智能终端这些方面从早期直到当前出现的种种安全问题，再次提醒大家目前安全威胁的广泛性和严重性。

抛开其具体内容不谈，质疑一直是黑客精神的基本基因之一。

值得一提的是，本次会议的赞助者除了启明星辰和绿盟这两家老牌的独立安全厂商外，多了腾讯的身影，因此抽奖的奖品也变成了身着盔甲的企鹅。互联网巨头在安全界的布局可谓无孔不入。

肖新光
网名江海客，安天实验室首席技术架构师，研究方向为反病毒和计算机犯罪取证等。微博：weibo.com/seak。

开放平台开发面面观

开发者和开放平台之间的矛盾，最核心的还是利益之间的冲突。

随着Zynga的成功上市和Instagram的火爆，基于开放平台的应用，越来越受到创业开发者和风险投资商的关注。但Zynga的成功上市和Instagram的成功卖出，并没有在开放平台和开发者之间起个好头并带来良性的互动，相反，开发者和开放平台之间的矛盾还在加剧。

我们经常会看到类似这样的新闻：社交游戏开发商不再开发Facebook游戏，例如CrowdStar是与Zynga齐名的社交游戏公司，2011年公司营收的90%来自Facebook，但它预计2012年公司收入的90%将来自移动设备游戏。这是因为Facebook在采取推广策略时并未一视同仁。如果一位Facebook用户接收了30名好友的活动状态更新，当其中一名好友开始玩Zynga的《Castleville》游戏时，该用户便有可能收到相应的通知。然而如果该好友所玩游戏并非由Zynga开发，这位用户便无法收到通知。这种状态通知是一种病毒营销，可以极大地影响应用的推广效果，吸引更多用户下载并参与游戏。但除Zynga外，Facebook已封锁了其他游戏的病毒营销窗口，导致开发者不得不在Facebook上投放更多广告。还有一种说法认为，虽然Facebook创造了一个繁荣的应用平台，但由于算法一直对外保密，而且经常调整，给应用开发者带来了苦恼。

虽有上述种种潜规则和不公平，多数开发者最后还是选择Facebook作为开发平台，是因为Twitter对待第三方开发者比Facebook更过分。Twitter近来关闭了很多API，包括禁止Instagram接入Twitter用户的粉丝列表等。其实，开发者和开放平台之间的矛盾，最核心的还是利益之间的冲突。

国外的开放平台是如此不公，那么基于国内开放平台的开发者的状况又是如何呢？我自己基于淘宝、新浪、腾讯、京东商城开放平台，也做了一年零八个月的开发，这期文章，我首先谈谈在淘宝开放平台上开发的感受，与对开放平台开发同样感兴趣的创业者与开发者共勉。

在开放平台成熟度、稳定度与商业模式上，淘宝是最清晰的。淘宝的Top开放平台，历经几年的磨炼以及像我这样的小白鼠的不断反馈，在技术架构和稳定性上，已达到了工业级开发和开放的程度。随着聚石塔计划的推出，参与其中的大卖家和第三方开发者，他们的服务器将和淘宝服务器放在同一个机房，这样可以有效避免下载速度慢、订单下载不全等问题。对参加聚划算、双十一、双十二等活动的大卖家来说，优势非常明显。

淘宝的痛还是在商业模式上。开放平台API最近几年不断调整和修改，使得开发者有点儿疲于奔命的感觉。例如2012年针对淘宝客的应用，淘宝做出了一个很艰难的决定：淘宝客应用和工具禁止提供淘宝商品搜索服务，这给淘宝客们带来极大的不便。一方面，淘宝从消费者隐私数据考虑，防止第三方应用滥用并进行不当的数据分析；另一方面，淘宝也不愿意将这些数据暴露，与第三方开放平台共享。在数据API、直通车API这两块最赚钱的应用上，淘宝对草根创业者彻底关闭了大门。

不过，我是做进销存应用的，我只想拿到与订单密切相关的一个数据：这些订单是从哪个通路过来的?我不需要全方位的数据平台，就只这样一个简单需求，淘宝也不开放。因此，我给淘宝的建议就是再对数据这样的核心API进行分级管理，把一个大API拆分成多个场景，由开发者申请自行组合。

邢波涛
北京新软孚信息技术有限公司技术负责人。关注SaaS管理软件和B2B、B2C电子商务的融合。

大数据时代的机遇与挑战

KDD 2012大会见闻

文 / 张驰原

ACM SIGKDD是数据挖掘领域的最高盛会，这次在北京国家会议中心举行的KDD 2012是第一次在亚洲地区举行，在收录论文数量和参会人数等各方面都创造了纪录，会议还引入了诸如暑期学校、"疯狂30秒"等新元素。为了拉近学术界和工业界之间联系的Industry/Government专场，以及为本次会议专设的Asia-Pacific专场也都是亮点。

时间地点人物

KDD 2012于8月12日-15日在北京国家会议中心举行，据主办方说，这个日子是经过精心挑选的：在2011年KDD结束之际，大家讨论来年的会议要在北京召开时，就开始担心天气和相关的自然灾害问题，经过各位数据挖掘大牛的人肉数据挖掘，最后定下了上面这个日子。最后看来结果相当不错，在闭幕大会上参会者还专门为此感谢了各位预报"专家"。

KDD 2012的注册人数超过一千，比以往任何一届都多，更有许多附近跑来旁听的人——因为除了午餐和晚宴，所有的会场都是不用任何证件就可以直接进入的。

起因、经过、结果

Big Data

本次大会的主题是Big Data，随着数据采集（各种移动、多媒体设备）和存储（巨大的网络空间，或者时髦一点的"云端"）的发展，我们所面临的数据越来越多也越来越"Big"。单从数量上来说就已明显超过了常规存储设备的限制，例如百度CEO李彦宏在Keynote上就提到，图片搜索里的数据量在很短时间内就超过了之前一直积累起来的传统网页搜索的数据量。另一方面，数据的复杂度也在不断增加，使得之前的简单模型变得不再适用，例如UIUC韩家炜在他的Keynote上介绍了他们组近来关注的异质网络上的数据挖掘问题。

"Big Data"到底和其他类似的表述，诸如"large-scale Data"、"Massive Data"有什么本质区别呢？事实上，对这个问题学术界并没有达成共识。大会的最后一天，主办方专门组织了一场讨论：Big Data到底指什么？它给我们带来了什么样的挑战和机遇？讨论会的形式是由主持人提出问题，然后由包括来自Berkeley、CMU、UIUC、北京大学、MSRA和ChoozOn的各位大牛嘉宾表述自己的观点，最后是嘉宾和观众的问答互动环节。

Michael Jordan激情演讲

事实上，可以发现各位嘉宾对这个问题也持有非常不同的观点。

Tutorial & Workshops

Tutorial和Workshop都安排在8月12日。前者一般是邀请某些领域的知名人士对一个领域做相对完整的介绍，而后者则多按主题展示一些新想法。

做口头报告时，由于演讲者的表达水平参差不齐，又受到时间限制，很多技术细节会被省略，如果你对议题非常了解，自然能够抓住关键，但对于刚入行的新手来说则很难。对于这类听众，Tutorial是一场会议里最有价值的部分了。听众能理解多少内容基本取决于与议题深度的差距，尽管Tutorial一般都邀请该领域的专家演讲，但内容由浅入深，对于刚入门的人也不会很困难。

Keynote

Keynote是一场学术会议中最盛大、最吸引人的部分。本次大会共有四个Keynote，第一个是百度CEO李彦宏带来的《9个需要研究人员帮助解决的难题》。但我个人觉得这场演讲不是特别吸引人，大概因为是第一个出场，并且标题也比较震撼，导致期望过高。

Big Data Panel的第一位发言者

第二个Keynote来自数据挖掘界的大牛，UIUC的韩家炜教授，他演讲的题目是《异质网络挖掘：新的前沿》。传统在网络上做挖掘考虑的一般都是同质网络，也就是网络上的所有节点都是同一类型。例如社交网络中一般所有节点都是用户，或者学术论文的共同署名网络所有节点都是论文作者。而异质网络则允许不同类型的节点同时存在于一个网络中，例如论文的作者节点会连接到论文节点，论文节点会连接到会议或者期刊节点等。这样的图结构往往更能反映实际问题中的数据结构，不过由于传统方法通常只考虑同质网络，因此需要发展全新的框架和算法来解决这个问题。韩家炜教授介绍了他们组在这个方向上的研究进展，包括在异质网络上的分类、排序等问题的解决方案，以及2012年7月刚出版的一本新书《Mining Heterogeneous Information Networks》。

接下来的两个Keynote分别来自学术界的两位大牛。首先是来自Berkeley的Michael Jordan（http://www.cs.berkeley.edu/~jordan/），他演讲的题目是《针对Big Data的分治和统计推断》，该主题的研究对象就是那种需要MapReduce集群计算和存储能力才能处理的海量数据。虽然MapReduce的排序、计数、索引等简单操作在工业界已得到广泛应用，但对于一些更复杂的统计学习算法，例如数据各个部分相互依赖无法有效地分离开来的情况，仍是难题。Jordan教授介绍了他们在这个方向上所取得的一些最新进展，其中的基本想法就是标题中所说的“分治”。不过，在这个方向上，仍然有很多难题有待解决，这一点在最后一天关于Big Data的讨论会上，Jordan教授再一次做了强调。

接下来是宾夕法尼亚大学的Michael Kearns（http://www.cis.upenn.edu/~mkearns/）。他是2010年图灵奖得主Leslie Valiant（http://people.seas.harvard.edu/~valiant/）的学生，早年和导师一起在PAC学习理论方面做出了许多贡献。他说这次被邀请做Keynote演讲，大概是因为最近几年所做的关于社会化计算的工作和KDD有许多关联。他演讲的题目是《社会化计算中的实验（及所产生的数据）》。社会化计算是指组成一个社会网络的人们共同完成一个计算任务。简单的例子是网络节点的着色问题，目标是使得所有节点的颜色一致，或者使得所有相邻的节点具有不同的颜色。我们知道，如果做全局优化，前者完全是平凡的问题，而后者则是极难的问题。然而这

里的社会化计算是一个非常特殊也更接近实际的模型：网络上每个人只能观察到自己和相邻人的状态，在这个限制下尝试解决问题。Kearns教授在试验中发现，要描述这类问题的复杂性，需要发展与全局优化完全不同的理论框架才行。例如相同颜色问题随着网络结构的不同可能会变得非常困难，而相邻点着不同色的问题却有可能会变得很容易解决，这无疑非常有趣。不过Kearns教授总结说，由于需要真人参与实验，因此，他们的实验一般都限制在30人以内，从这一点看，倒是和“Big Data”差别挺大。

演讲和海报——广告时间

虽然Tutorial和Keynote都很精彩，但一场学术会议的主体还是论文演讲和海报展示。KDD 2012给所有录用的论文都同时提供了口头演讲和海报展示，并且还提供了免费打印海报服务。

论文演讲是许多专题同时进行的，对听众来说需要好好选择。每位演讲者有20分钟的演讲时间，之后是简短的提问环节，非常紧凑。演讲人的表达能力参差不齐，对于不善言辞的人来说，海报展示起到了很好的辅助作用。

海报展示环节是在晚宴大厅进行的。三个小时的时间，整个大厅摆了各种白天做过演讲的论文海报，大家一边吃东西一边讨论。论文的作者大多会站在自己海报旁给路过的人讲解。让更多人知道自己的工作，我想就是论文的作者在此时的主要任务吧。

学术界与开源社区很相似：论文的数量越来越多，但除了几项核心的工作，其他工作在重要性和影响力方面大同小异，同类的工作通常也很多，互有优劣，很少有一枝独秀的。因此，让更多人知道自己的工作才显得尤为重要，因为学术影响力就是通过别人参与你的工作来建立起的。虽然听一个云里雾里的演讲，或者在海报旁逗留5分钟的讨论似乎很快就会忘记，但说不定将来会有大作用。我们常会碰到这样的情况——知道某件事物的名字后才发现原来身边比比皆是。

对听众而言，当然也很有乐趣。我在会场里来回逛了好几圈，在各种海报前搭讪作者或者被作者搭讪，其中有不少有趣的工作。特别是，如果在论文演讲中由于时间关系没有能问到问题的，这时就可以找到解答了。对论文的作者来说，这三个小时的时间要比演讲累得多，而且还没法围观其他海报。

疯狂30秒

“疯狂30秒”是2011年的新元素，每位论文作者可以提交一个最多不超过30秒的视频，介绍他们的工作。视频会于每个Keynote开始之前在大厅的投影屏幕上播放。

这是一个展示个性的舞台，有的类似幻灯片，有画面没声音；也有的用“疯狂”的声音在飞快地念完一段无法识别的文字（给人感觉是30秒读了一遍论文）；还有些视频做得非常专业。这个环节给人留下了深刻的印象，在大会结束时还评出了最佳视频奖。

午餐和晚宴——社交时间

主会的第二天，碰到了来做演讲的好友Ming Ji，也是托Ming的福我在这一天结识了不少圈内人士，午饭时了解到一些有趣的学术圈故事，还有这次KDD主办方所遇到的一些困难。社交应该也是会议的另一大目的吧，因为“协作”是科研中的重要环节，所以社交也就理所当然地成为研究工作中不可缺少的一部分了。

张驰原

毕业于浙江大学，致力于机器学习相关研究，现于美国麻省理工学院计算机科学与人工智能实验室（CSAIL）攻读博士学位。热爱并积极参与开源软件的开发与推广。

OpenStack：云时代的 Linux？

首届OpenStack亚太技术大会侧记

文 / 林溪

2012年8月10日-11日，首届OpenStack亚太技术大会（OSAC）在北京和上海两地同时召开。两天时间内，北京会场有1600多人次参会，上海会场有近700人次。一个还非常年轻的开源项目能有如此人气，的确超乎预期。同时，会议的成功还有另一层含义：这是一次完全由社区驱动的技术盛会，从演讲者的邀请和遴选、会议组织到赞助，都由来自CSDN和COSUG（中国OpenStack用户组）的社区成员分别负责。

充分开放：OpenStack崛起的原因

OpenStack诞生不过两年，却异军突起，在支持厂商（NTT和KT等电信运营商，众多云计算公司，Intel、HP、思科等传统IT厂商，也包括Yahoo!、新浪等互联网企业）、社区活跃度和发展势头等方面都很快超过了技术上更成熟的许多其他同辈。其中过程和背后的原因，值得平台技术和产品的决策者、研发者好好思考。

毕竟，在IaaS开源解决方案上的竞争非常激烈，Citrix捐献给Apache的CloudStack，与AWS兼容的Eucalyptus、Nimbus，早在2008年就开源的OpenNebula，Red Hat的CloudForms等，都有自己的特色和拥趸。OpenStack的成功仅仅用开源来解释，不通。

NASA和Rackspace的背景当然是OpenStack最初迅速引起广泛关注的原因。但技术上的因素同样存在，新浪SAE技术经理程辉就表示，当初之所以选择OpenStack，主要原因是感觉架构上不错，代码质量高，而且编程语言是团队成员喜欢的Python。

与其他开源项目相比，充分的开放和活跃的社区才是OpenStack的最强武器。这与许可证采用对商业版本友好的Apache 2.0，研发方向都真正由社区而非某个公司主导，社区本身的加入和各种信息也完全开放等密切相关。与会很多专家回忆自己最初选择或者转向OpenStack的原因时，都提到社区的活跃让人看到了希望所在。

而正在积极筹备中的OpenStack基金会将从制度上真正将责任和控制权转给代表社区的中立机构，使各大厂商能打消顾虑，这将为项目的长久发展打下坚实基础。本届大会上，社区代表、Mirantis联合创始人Boris Renski介绍了基金会的最新进展，原政策委员会已成为技术委员会，负责技术层面的管理，采取独立、精英式的模式。同时成立董事会负责机构的管理和运营。

应用与实践

本届大会上，数十位国内外技术专家分享了各领域的实战经验。而来自国内的应用与实践更引人注目。

新浪云计算在今年6月份正式发布了国内首个基于OpenStack的IaaS公有云产品——Sina Web Services（SWS）。加上SAE，目前国内最大的PaaS平台，还有新浪云商店这个面向普通用户的SaaS层面的产品，新浪目前已经基本完成云计算三个层面的产品布局。本届大会上，程辉还披露了SWS的一些技术细节，比如公开了SWS网络部署拓扑，介绍了OpenStack安全上的一些改进，还提到了“具有中国特色的”负载均衡方案的架构和设计以及OpenStack Swift在新浪的部署情况。

京东商城虚拟化工程师何斌的演讲引起了很多

参会者的关注。他介绍了OpenStack在京东商城的应用实践，例如开始尝试在OpenStack平台接入部分线上业务、实现OpenStack自动化部署、实现桌面云（已交付给呼叫中心试用），还提到京东正在开发的基于OpenStack Elastic Scaling和ELB的产品。在今年京东店庆促销时，这些应用帮助京东战胜了用户暴增的挑战，为用户提供了平稳的服务和良好的用户体验。他透露，京东在内部实现桌面云项目时，对OpenStack做了很多二次开发。

网易杭州研究院副总监汪源为大家分享了网易在生产环境中应用OpenStack的一些问题与经验。目前有几十个节点，支持3个线上产品，包括网易相册和网易邮箱人脸识别都完全部署在OpenStack环境中。但没有使用Swift，因为他们自己的对象存储系统更加成熟。

趣游集团云研发项目负责人刘继伟在此次大会上隆重介绍了OpenStack在趣游的应用。他说，趣游从2011年11月16日开始部署OpenStack，重新实现了Openstack的功能，例如控制台、监控和升级等基本功能。

机遇与挑战

新浪的程辉在其演讲说，OpenStack是Amazon Web Services（AWS）的开源对应，是一次历史机遇。在美国，基于OpenStack的商业模式正在形成，涌现出了一批基于OpenStack的创业公司。Piston提供Piston Enterprise OS及企业服务，特点是快速部署和安装，只需一个U盘便可在几分钟内将一批裸机架部署完成；Mirantis提供OpenStack培训和人力资源服务；SwiftStack提供基于Swift的私有云存储服务解决方案；Nicira利用OpenFlow协议和Open vSwitch为OpenStack提供商业的网络虚拟化解决方案。中国目前还没有这样的公司，这几个方向都是发展机会。另外，目前国内公有云有谁会成为中国的AWS一直是业内人士在关注和讨论的问题。OpenStack的解决方案将会为这场竞争提供很大的助力。

的确，OpenStack在不少方面都与Linux有相似的

首届OpenStack亚太技术大会在北京和上海两地同时召开，会场爆满

感觉。Linux当年与FreeBSD等各种BSD开源版本相比，最初的底子也很薄，但靠开放、活跃的社区，得到更多厂商支持，在各个领域攻城拔寨，最终全面胜出。OpenStack能否重复历史呢？

Linux的成就很大程度上要归功于Linus Torvalds本人，他在开放与控制之间做到了很好的平衡。而他的长期领导保证了Linux的发展方向的稳定。而目前OpenStack缺乏这样清晰的领导者。项目原始的创始方之一NASA实际上已经隐退，最初的创始人中不少都分别自组公司（Nebula和Piston等），从事商业支持去了。而众多厂商目前的热情，其实也隐藏着很多利益之争。

更重要的一点是，Linux是击败了众多强劲商业产品而成为时代王者的。这个任务对于OpenStack更加艰巨，毕竟Amazon现在已经非常强大。与Amazon的封闭、专有开发模式相比，开源模式能否再次在速度和创新上战而胜之，极具观赏性。

OpenStack对中国而言也是一次机遇，正因为OpenStack很年轻，自身也有缺陷，给已经在能力和人员上发展壮大的中国技术界贡献自己力量，并与之一起成长提供了许多机会。但程辉也表示，国内OpenStack方面的人才还是非常缺乏，这将成为OpenStack在中国发展的一个阻碍。他呼唤更多的技术人员关注和参与进来。如果OpenStack真的成为云时代的Linux，这将是共同创造历史的机会！

梁捷
UC（优视科技）技术总裁。1998年毕业于华南理工大学计算机专业，后长期耕耘于中国的电信和互联网市场，在电信及网络计算领域拥有超过10年的技术研发和管理经验。

梁捷专栏

让Web App来得更快一些

在刚刚结束的2012HTML5峰会浏览器专场上，我和各位业界的同仁分享了我们观察到的Web App的一些趋势和UC做的工作。

Web App规模化很快到来

2011年9月，UC推出了国内首家移动Web App应用中心，当时只是收录了很少的Web App，主要是一些自己做的实验性产品和一些合作伙伴的产品。后来我们一直在对UC应用中心进行改进和升级，比如提供应用搜索、添加应用图标到桌面书签、通过UC的账号管理自己的应用等。

经历了十个月左右的发展，目前UC应用中心HTML5应用收录数量已经超过400款，活跃用户已经超过3000万，添加应用的数量超过9000万（“添加”相当于App Store的“下载”）。这样的一个成绩相对于App Store和Google Play还是有不小的差距，但可以很明显地发现，用户对于Web App有着旺盛的需求，尤其对使用游戏、信息阅读、社交和生活工具等类型的Web App有着很高的热情。

再看这批最早的HTML5应用开发者类型，他们大多都是从传统的Web网页、网站转型而来的，在前端技术方面有着比较好的积累。新的HTML5应用大多集中在浏览和阅读，以及工具和垂直类领域。不过还有不少的开发者是专门做HTML5游戏的，目前在UC平台上就有一款名为《新佣兵三国》的HTML5游戏，每个月的营收已经超过50万元人民币。

从技术层面看，Web App无疑可以让整个的应用开发成本变得更低。从产品层面看，Web App的互通性是非常优越的，营销和推广成本也都很低，而且这个互联互通的情况是属于不可逆的，一旦大家习惯了就不会再往回走。从我们的分析来看，随着标准的完善以及开发工具的升级，2013年，Web App将会有一个爆发性的发展。不过我们也都很清楚，Web App市场环境现在也有不少的问题，比如性能、兼容性不够好，W3C标准尚未完稿等。但这些都是可以通过努力解决的问题。

打造最好的Web App平台

2011年6月，UC正式推出了自主研发的U3内核。U3内核是从2008年开始立项，利用Webkit技术做深度改造和发展的一个产品。也许有朋友会问，同样是基于Webkit，什么样的浏览器才是真的具有独立内核的?

从我的角度来看，所谓的独立内核，是要看这个内核是不是独立发展的一个产品线，是不是在这个内核上具有创新能力。U3内核在推出

后一直都是自己独立发展的，不仅仅只在客户端上面有很多创新的新体验，而且还把UC独创的云端架构的一些优势集合了进来，从而打造了目前移动领域最优秀的Web App平台。

U3内核对HTML5特性支持良好，基本上一些比较常用的HTML5的特性都得到支持，Canvas、WebGL、WebSocket、视频、音频以及定位、方向等。同时，我们一直致力于对运行效率的改进，例如在最近的几个版本中，UC浏览器的相关性能测试在快速提升，比如JS性能、V8测试和Sunspider都有长足进步。8.6版本我们做了渲染架构的优化工作，使得HTML5的动画支持效率也得到了大幅的提升，speed reading测试的帧率从14提升到20，而Camera 3D的帧率从11提升到18。从这些数字我们可以看到U3内核始终在独立发展并且每一个新版本的发布都会有新的进展。

目前UC还在致力于解决一些比较核心难度又相对比较大的门槛性的问题，比如在过去一年多的时间里，UC已经分别跟支付宝、中国银联进行了合作，率先提供了在手机上网的安全支付渠道，解决了移动互联网商业化过程中的“造血”问题。

另外我们还会让U3内核覆盖更多的平台，目前U3已经支持的平台包括Android手机、iPad、MeeGo。我们计划用一年时间使之支持iPhone，Android平板还有WinPhone。未来长远一点看，我们不仅仅是希望在这几大主流智能平台实现对HTML5的支持，其他的平台比如MTK等，也希望能够达成同样的目标，真正帮助开发者实现“一次开发、多平台运行”。

打造开发者社区，助力行业规模化

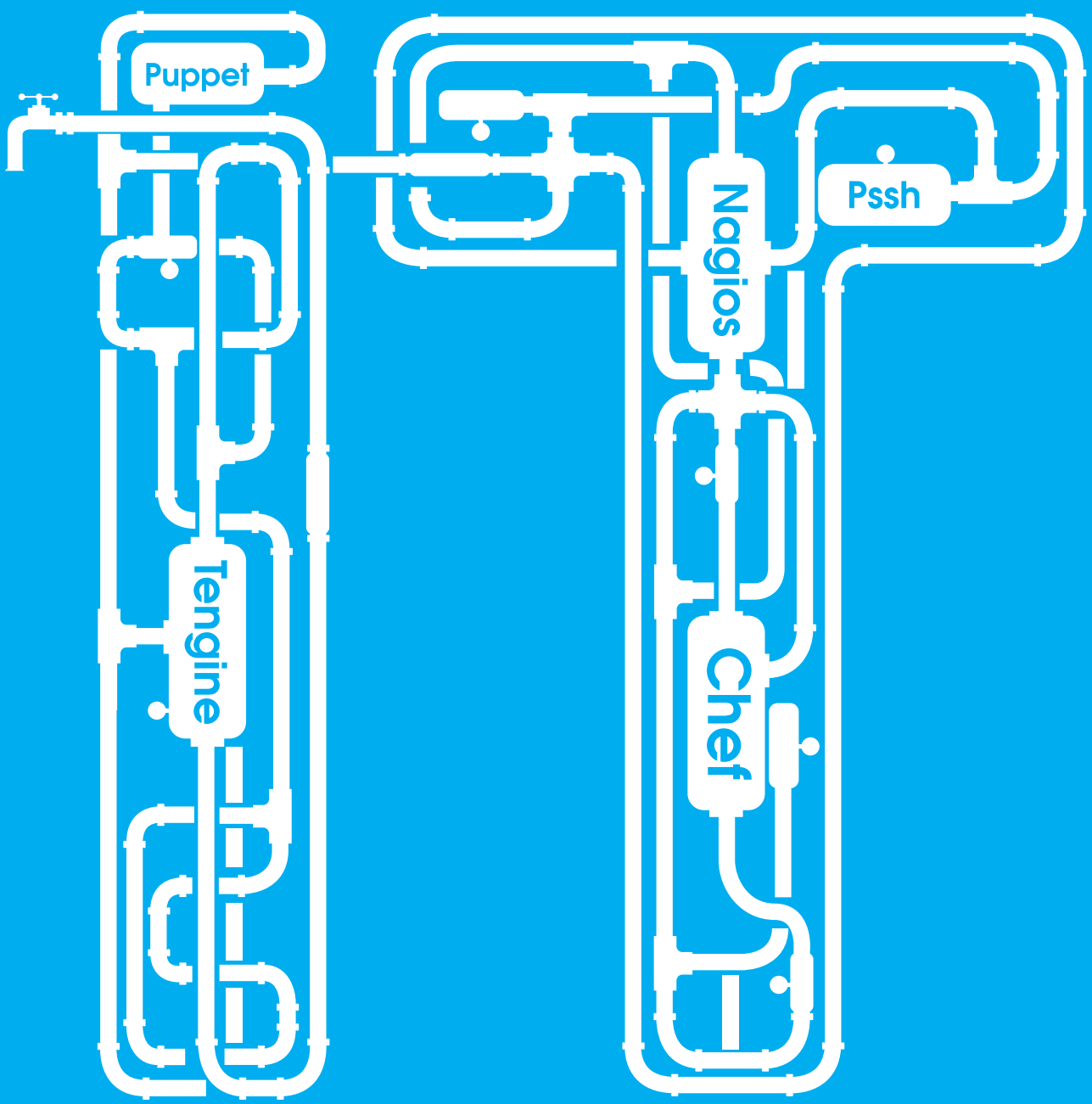
现在UC拥有也许是全国最大的一支移动互联网研发团队，已有超过1300个员工，其中80%从事产品和研发。在UC这个平台已经积累了很多优秀的人才和经验，于是我们想建立一个对外的平台，进行资源分享。我们计划建立一个Web App开发者社区，这个社区的工作不仅仅是提供一些资料，更重要的是建立浏览器团队和Web App开发者之间沟通的平台，从实操性的角度，去解决当前Web App开发者所面临的问题。

分享一个案例，不久前有一个开发团队发现他们开发的Web App，在首次打开时需要加载一个资源包，由于加载的过程过于漫长，使得用户的流失很大。于是他们就建议UC，希望我们能做一个安装的标准，使得这个资源包的加载可以在安装的过程中完成。这样用户在首次使用时，打开的速度就会非常快，就能够减少用户的存活率流失的状况。这就是一个我们希望开发者社区能够起到的作用的典型例子，我们希望通过这个平台，去解决大家遇到的问题。

目前的UC开发者中心还刚刚建立起来，仅仅提供了一些支撑工具，和一些与UC结合开发的相关信息，未来我们会对其加以完善。目前提供的功能，比如UC浏览器的开放参数，获取方式很简单。我们只需要在URL里面加上关键字uc_param_str，开发者就能够在用户下一次点击这个链接时发出的请求里面，获得UC独有的一些参数。比如通过up参数，我们可以知道用户的联网方式，了解他到底是cmwap、cmnet还是WiFi接入网络，如果是WiFi的用户，我们可以给他更好的体验、高分辨率的图片等，帮助我们的开发者们提供更好的交互体验。

未来我们还会推出一个基于UC浏览器的开发调试工具，用这个工具，我们可以用UC浏览器去看网页上的一个项目。将手机通过数据线或者WiFi连接到PC上面之后，开发者可以在PC的大屏幕上去监视自己开发的Web App的运行状况，比如可以看到它的DOM状况、进行JavaScript的断点设置调试、使用CSS的调试器等。

目前我们内部已经在测试这个开发调试的工具，内部评价较高，应该可以将开发效率提高不少。我们应该很快就能推出，敬请各位开发者关注和期待。


Puppet
Tengine
Nagios
Chef
Pssh

IT运维自动化

我们依赖于互联网的同时，互联网也依赖于我们。

设计、构建和维护一个用户规模不断增长的网站，在其进入系统管理和软件开发阶段后，将面临许多不同寻常的挑战。

一方面，服务器不眠不休，因而不存在“适宜”的时间来对它进行更新和维护。另一方面，应用变得越来越复杂，除了更多的部件要构建与维护，问题的出现方式也越来越多样化。人、设备会以我们能够想象到的所有方式发起挑战，然后在想不到的方面让我们目瞪口呆。在这种情况下，保证应用稳定、高效运行全仰仗一个特殊的工程领域——IT运维。

IT基础架构越来越快速的发展，令企业IT基础架构规模不断扩张，企业往往将面临短时间部署大量服务器的要求，传统的运维手段在此时已捉襟见肘。经过十数年的发展，IT运维原本依赖的手工重复操作和脚本，已逐渐发展为成熟的工具。对运维而言，自动化是必然的方向，它解放了运维工程师的双手，但同时也意味着要学习新工具，在新环境下积累经验。

本期封面报道将从IT自动化的出现和发展谈起，讲述从小型创业公司到大型互联网企业的最佳实践，角度涵盖性能监控到软件部署，介绍最新的IT自动化运维工具和应用，为不同背景关注IT运维的读者提供参考。

从时代变化与规模谈自动化运维

文 / 余沛

时代变化所引起的运维视角不同

在计算机应用的发展历史中，运维工作一直是必不可少的重要环节。无论在什么年代、什么场景，保证服务的正常可持续运行都是运维的最终目标。但在不同时期，运维实施的手段、关注的重点却有所不同。

在计算机应用的早期，计算机仍然是一台昂贵且娇气的设备，那时非但应用场景远没有今天这样广泛，网络体系也没有今天这样的复杂和敏感，电子技术也远不如今天成熟。在那个时期，运维关注的重点通常以硬件为主。除了计算机本身的硬件，与之相关的电力及其他配套设备也在范畴之内。

到了中期，计算机由大、中型转向微型化，局域网络发展迅猛，金融、通信、电力、航空、学校等各类商用、民用场景的快速跟进，不但使得计算机的总体使用量大增，而且单位范围内需要管理的计算机规模也在不断加大和复杂化。在这个过程中，形成了以设备管理（包括网络设备、操作系统管理）、数据存储与容灾、目录与内容管理、资产管理和信息安全管理为主的、一体化的早期运维体系。

再到后来，互联网模式占据了最新潮流且至今未衰，科技和业务模式带来的变化，也催生整个运维体系的转变。早期一体化的运维系统被更加明确与细致的分工所拆散。IDC代替了自建的局域网小机房，使得通信主干的运维工作从原有体系中剥离；规模的爆炸式扩大也使得即使同一IDC内的局域网管理也更加复杂；安全的挑战越来越严重，使得传统基于局域网粗暴隔离思想的安全管理无法满足开放互联网时代的要求。同时，人们也发现一些运维事务简单地随规模增长而线性增加（如资产管理、自动装机等），本身不具有复杂度的变化，而另一些与业务相关的事务，则随着规模增长复杂度成倍增加。

由于这些原因，使得运维工作开始明确分工，逐项剥离。系统、硬件、网络衍生出专职的系统管理员（SYS）职位，安全也开始从运维独立，资产管理、自动装机等工作不再是运维工程师关注的重点。相反，与业务可用性紧密关联的调度、监控、关系管理变得更加上层。

规模变化所引起的运维手段不同

在互联网时代以运维的角度来看，对于不同规模、场景的应用，运维所处的位置、发展的历程也有一定差异。

百台以下规模的运维

一般来说，小公司有几台到数十台机器的规模，运维主要关注的点还是在系统及应用服务的稳定性上，不一定有专门的运维人员，通常是RD自行维护所开发的系统。不会去系统化地梳理各项运维指标，而是以人的经验为主，来判断和设定系统正常与否的标准。

在这种场景下，运维工作通常扮演的是“救火队员”角色，一旦出故障，运维人员才开始跟进、解

决。故障解决也基本是Case by Case的形式，不太会有流程化的总结和问题库、知识库。其运维方式无非是靠手工，辅以一些自行编写的小脚本。这种方式虽较为原始，但对许多初创公司来说，也算是一种低成本下可灵活实施的手法了。

在这个阶段，一些小巧的工具显得很有特色，例如在系统管理方面，能够同时对多台机器进行操作的Pssh（http://code.google.com/p/parallel-ssh/）、OmniTTY（http://omnitty.sourceforge.net/）等，在数据库管理方面，phpMyAdmin等足以应付平时的工作。

千台规模的运维

当规模增长到一定程度，依赖手工管理自然已无力应对。许多互联网公司的服务器早已跨入几百甚至千台规模，脚本化、批量化管理占据非常大的比例。

同时，在这种规模下，按垂直划分的运维工具也开始大量应用，无论是自行开发的还是利用现有开源软件，针对某一特定领域的管理系统显得尤为重要。

在这个阶段，运维主要精力放在监控（采集、报警、展现图表）、部署上线（配置管理）、数据备份方面，因为机器数量庞大，所以集中式的操作平台是必备的，针对不同的领域，也有相当成熟的开源软件可以直接使用。

在监控方面，Nagois和Catci应用最为广泛。

作为一款开源的免费网络监视工具，Nagios能有效监控Windows、Linux和Unix的主机状态、交换机路由器等网络设置。在系统或服务状态异常时发出邮件或短信报警，在状态恢复后发出正常的邮件或短信通知（如图1所示）。

Nagios的特点包括以下几方面。

- 可以通过多种协议对目标服务器进行信息采集（SMTP、POP3、HTTP、NNTP、PING等），避免在目标服务器安装客户端带来的风险。
- 本身体系非常开放，可以自定义编写采集脚本以监控系统、应用的状态。简单的插件设计使得用户可以方便地扩展自己服务的检测方法。
- 具有并行服务检查机制，同时具备定义网络分层结构的能力，用“Parent”主机定义来表达网络主机间的关系，这种关系可被用来发现和明晰主机宕机或不可达状态。
- 当服务或主机问题产生与解决时将告警发送给联系人（通过Email、短信、用户定义方式）。
- 可以定义一些处理程序，使之能够在服务或者主机发生故障时起到预防作用。
- 自动的日志滚动功能。
- 可以支持并实现对主机的冗余监控。
- 友好的Web界面用于查看当前的网络状态、通知和故障历史、日志文件等。

Nagios和Cacti是经常配合在一起使用的监控系统，Nagios适合监视大量服务器上的大批服务是否正常，重点并不在图形化的监控，其集成的很多功能例如报警都是Cacti没有或者很弱的；Cacti主要用途还是用来收集历史数据和画图，因此界面比Nagios漂亮很多。

而在部署方面，Puppet是用于大规模集群管理的神器。其本身使用Ruby语言开发，基于C/S架构

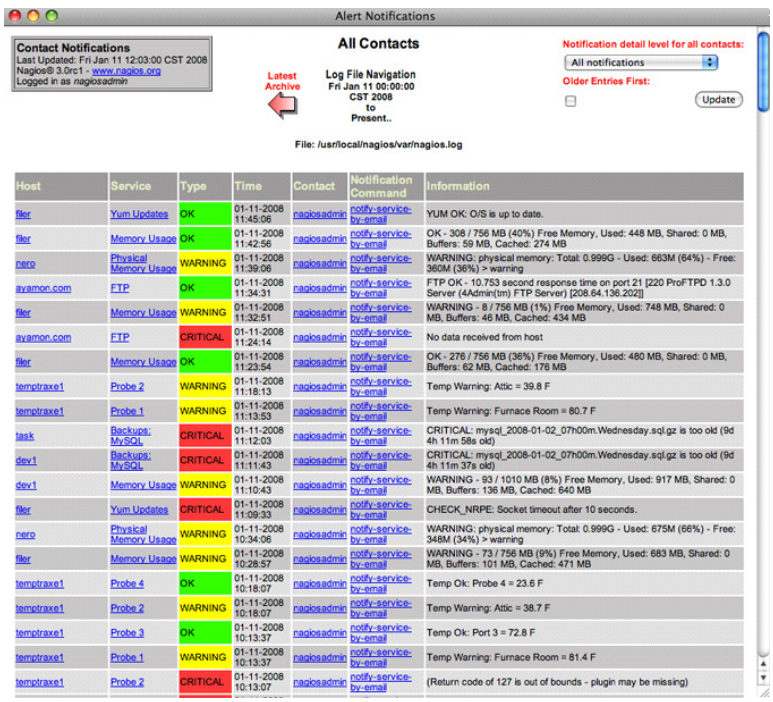


图1 Nagios报警通知界面

（如图2所示）。在每台机器上部署的客户端每隔一个指定的时间会连接到Master检查资源变化情况，若资源发生变化，将按配置动作进行相应的操作。

Puppet将所有可操作对象抽象为资源，目前涵盖了40多种，如：File、User、Group、Host、Package、Service、Cron、Exec等。

Puppet通过抽象资源的方式，使得每台机器能够“清楚”其本身“应该”是什么“状态”，而客户端根据当前是否达到这个状态决定采取指定的动作。这使得Puppet不仅可用于传统的应用部署，而且通过合理的手段，也能够将比应用部署更频繁的配置管理一并解决。甚至可以在Master端外接自己开发的平台，通过集中配置方式管理各项“资源”，实现高度灵活的自动化管理体系。

这类垂直管理系统的使用及活跃，极大减轻了运维人员在重复性、批量化操作方面的负担，能够非常有效地在各自领域完成既定的运维子目标。但其缺陷在于只能针对某一垂直领域的特定问题进行高效处理，对于它们之间的关联性很难应对。因为运维的本质是保证服务的可用性，而自动化运维则是在完全保证这一前提下，尽可能将需要人干涉的部分处理掉，所以判断其优劣的标准则是——与人工处理比，对服务的保证有没有提高。如果仅是解决报警、部署这些单一动作，后续仍然需要人去处理、去关注、去判断的话，就离这个目标还有距离，谈不上真正的自动化，只能算是工具化。

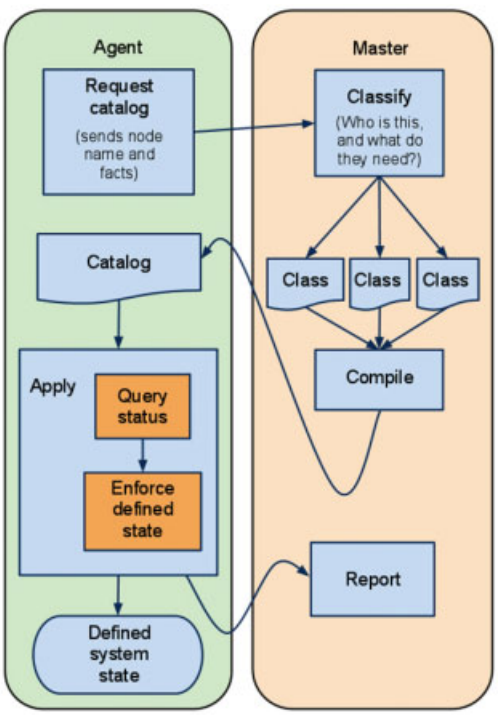


图2 Puppet Master–Agent结构图

上万台规模的运维

对于规模已达到上万台、十万台的各互联网巨头来说，其业务间交织纵横，甚至某一单一业务内部可能也已非常复杂。传统的开源软件大多已无法覆盖这样场景下的运维工作。另外，在这个阶段，除了各服务自身的监控信息、部署信息等需要运维外，服务与服务间的关联关系、上下游变更所带来的影响、业务的串联也显得尤为重要，它们经常是牵一发而动全身。互联网巨头都在根据自身的业务特点，纷纷自行研发成体系的自动化运维平台。在这个阶段，各公司更关注各平台之间的联动性，更关注运维的本质——即真正的自动化：不仅是自动发现问题，更要能自动协助解决问题，以保证服务的稳定。

各大公司自主研发的历程，是由单一的任务解决向平台化过渡，运维关注的重点也由可用性发展到易用性、灵活性，最终实现自动容灾，以及资源可伸缩调度。

以Google为例，能够将URL（统一资源定位）的概念用于运维体系，将每一种服务（无论对内、对外）都抽象为一种资源，提供这种资源的服务将自身信息写入，使用方通过URL来得到资源的实际位置，当资源发生变化时，使用资源的一方能够感知，并根据自身业务做出相应的动作。

国内的互联网公司也有不少家正在往这个方向追赶。这些自主研发的智能运维平台，除了像开源软件那样能满足常规的监控、部署备份等需求外，更能站在“服务”的角度关注其最终状态。

例如，某项服务有N台机器提供负载均衡，会向某个状态池集中写入其状态，相关的监控通过收集

这些状态，按照指定的配置规则进行动态监控。一旦发现状态异常，如负载过高时，能够触发部署动作，从资源池中再拉几台机器进行部署。部署信息都在状态池中，该安装什么软件、什么版本、如何部署都由系统自动完成。完成部署后，新机器上的服务被启动，启动时再向状态池中注册自身，负责负载均衡的设备感知到关注的状态信息变化（有新机器加入）后，刷新配置将新机器的服务纳入对外服务列表。

自动化运维与基础架构之间的关系

随着运维技术的进步以及运维体系的完善，自动化运维作为一个既不算崭新却又充满活力的方向，也随着规模、场景的变迁迎来新的挑战和变化。运维的活动范围，更多介于硬件与操作系统之上、应用之下。其与基础架构之间也像是人的两条腿，相辅相成，总是一前一后交替往前推进。基础架构决定运维方向，同样运维体系又使得基础架构发挥最大收益。

故而自动化运维平台的根本，不是说仅仅去把操作界面化、OA化，让人们简单地在Web上点击按钮就能管理系统。而是在底层的基础架构与上层的业务系统之间搭建一个良好的桥梁，使得业务系统能够充分、稳定却又不必过度关注底层架构特性。

这时的运维，已不再仅是消除故障、打扫设备的后置服务，而是能够在业务开发时期介入、伴随整个业务共同运行的一种特殊服务。以Google的Borg为例，通过引入一套标准库，按照指定的规范编写应用，使得编写出来的应用本身就能满足对应基础架构下的可靠运维，无论是统一的运维状态接口，还是灾备、自动缩扩容，以及变更时的关系调整，都能够很好地应对。

云计算带来的变革与挑战

虚拟化、云计算的发展，又使得运维工作产生了进一步的变化。中小公司不必再考虑诸如容灾、备份方面的事宜，资源的按需交易不仅使得资源不再浪费，也使得业务调整时的伸缩变得更加容易且经济上更加划算，的确大大简化了传统意义上的运维工作，是未来中小互联网公司发展的重要趋势。这时，运维工作的重点将转移到平台架构的选型与优化上来，运维需要更关注业务特性及与之相关的技术体系，帮助研发决定各类云服务的选型、评估其对业务的适用性。

在可预见的未来，运维的角色将变得越来越重要，这种重要性的提升关键在于运维在整个技术体系中的参与度及所处位置发生的变化。自动化运维的兴起，将传统意义上处于后置服务的运维人员带到了服务前沿，以往简单的追查故障、保障服务虽然仍占据运维工作的一部分，但随着自动化运维技术的发展，运维人员有更多精力、条件，投入到整个服务架构的梳理、设计中，甚至以提供基础组件的方式参与到研发过程，使得产品天生具有较高的可运维性。研发要关注运维，运维要有能力介入研发，具有对等能力的角色配合，各自负责不同的领域方向，这是技术发展大体系下的必然分工，也是运维摆脱苦力劳作，继续往更深技术发展的必然之路。

余沛

艺龙网平台总监。原百度高级工程师，专注于分布式及自动化运维方向，曾参与百度自动化运维平台核心构建，现负责艺龙网基础平台及自动化控制系统。

基础架构与产品运维并重

天涯网站运维实践

文 / 周小军

天涯社区网站拥有七千万注册会员，百万级的在线用户，日发帖量几十万。截止2012年8月，天涯在Alexa上的国内排名为22位，全球排名为119位。

天涯最基础、访问量最大的产品是论坛。围绕论坛，天涯还相继上线了博客、相册、天涯客、部落、问答、天涯微博等其他相关产品。天涯从最初的几百用户在线发展到百万用户在线，运维团队一直以来都在幕后支撑产品的发展。天涯的成长也是运维团队成长的历史，天涯运维团队的成长可以用三个阶段来概括。

阶段一 论坛架构扩展

天涯是个动态内容网站，所有的内容直接从数据库产生，最早的天涯论坛是IIS+MSSQL Server的二层结构，通过一台服务器来支撑，ASP应用通过ODBC直接访问后台数据库。天涯论坛页面还有一个特点，就是一帖到底。这不像普通的论坛按固定的回复数，例如20个回复来决定分页，天涯将主帖和回复全部展示在一个页面中。这样虽然对用户的阅读体验友好，但无疑加大了对应用和数据库的负载。

用户每访问一个帖子就会访问应用服务器，并在数据库端产生多个数据库事务，然后在应用服务器的内存中拼装成页面发送给用户，非常消耗应用服务器的CPU、内存以及数据库I/O资源。因此面对大量的页面请求，天涯应用和数据库服务器压力非常大，在高峰期经常出现IIS抛出500.13错误，或者频繁重启释放内存。应用服务器在高峰期CPU使用率经常是100%。

那时的天涯运维团队只有一两人，面对访问量一大就瘫痪的情况，运维团队开始和研发团队一起解决论坛的架构扩展问题。首先在前端部署页面级缓存以减少对应用和数据库服务器的直接访问。我们使用Squid做页面缓存服务。Squid是一个优秀的开源代理和Web缓存服务，我们将访问量较大的帖子和帖子列表等页面通过ASP函数生成HTTP过期标记。过期时间根据业务的特点来设置，例如帖子页和列表页的过期时间设置为5分钟。并且当用户有回复动作时，我们在回复成功后从应用端发送一条PURGE请求给Squid服务器组，通知其在缓存中删除指定的缓存对象，避免用户看不到最新回复的帖子。

在Squid初上线时，缓存命中率仅为30%~40%，于是我们又在Squid前端加了一层反向代理。我们使用了HAProxy代理服务，以URL Hash的方式将相同的URL导向各组Squid来响应，避免页面对象在不同的Squid中随机分布。通过这样的策略，最终命中率提升到60%~70%。

在数据库扩展方面，我们用最简便的办法——分库。原来的板块是一张表，分库后访问量大的板块都是一个独立的库，甚至由一组独立的数据库服务器承载。应用层通过数据接口服务来调用数据，使得应用层不关心数据库如何分布。早期的数据库平台是Microsoft SQL Server，我们用事务复制的方式来实现双机，其中主数据库服务器充当发布和分发服务器角色，从数据库服务器为订

阅服务器角色。在这个结构里，主数据库服务器负责数据写入，从数据库服务器负责数据读取。

天涯后期数据库迁移到MySQL后，数据库服务器采用常见的MySQL集群方式，每组数据库服务器由一台或两台Master和多台Slave组成。Master负责写，Slave负责读。

最终天涯的架构从二层扩展到代理、页面缓存、应用、接口和数据库五层，采用DNS来内部寻址，每一层都基本支持水平扩展。

在系统监控上，我们使用开源的Nagios来监控主机和网络设备运行状态，使用Cacti通过SNMP协议获取主机和网络设备的CPU、内存及网络基本性能。我们还针对Windows服务器使用C#语言写了一些监测工具，获得IIS和Microsoft SQL Server服务的状态数据，及时了解服务的运行状况。

我们使用IIS的Error重定向功能，将出错的应用信息打包成XML消息，发送到RPCXML采集服务器，用来监测应用错误、代码Bug和SQL注入攻击等问题。

这个阶段的天涯运维工程师身兼数职，不仅熟悉系统及架构，而且还得熟悉开发和产品，以保证系统在流量高速成长的情况下性能得以平滑扩展，使得天涯得以从百万级PV支撑到千万级PV。

阶段二 平台转型

随着对开源技术的不断深入应用，天涯的应用架构开始从ASP转向JSP，从IIS+Microsoft SQL Server+Windows平台全面转向Resin+MySQL+Linux的开源平台，此时的运维团队开始第二个阶段——平台转型阶段。

这个阶段的运维团队扩充到六七人，运维划分成三个组，分别负责系统、网络和数据库。每个组由各组主管负责本组工作。简单的运维流程和管理规范也随之成熟和规范起来。

由于天涯多是动态内容，不适合采用CDN方式来解决互联互通，因此我们采用双线接入的方式。运营商给分配了电信、联通双线双IP，为了减少业务的复杂性，我们没有把应用服务器划分为电信应用和联通应用。而是在F5-LTM负载均衡服务器上设置了电信和联通双IP，创建电信和联通虚拟服务器（Virtual Server），把用户请求统一转发到后端的应用服务器。

针对N-PATH模式的应用，我们用了二组电信HAProxy和联通HAProxy来分配LOOP IP。HAProxy后部署不同的应用服务器组，在HAProxy中通过URL拆分来转发指定请求到各自的应用服务器组中。

我们使用F5-LTM来处理众多小产品的负载均衡，使用LVS来处理高并发、大流量、单一产品的负载均衡，例如静态资源等应用。这样的配置方式是节省设备投入，减少运维成本。

我们采用智能DNS方式解决用户的线路判断问题。在DNS服务器上采集请求的DNS服务器IP，通过IP地址库分析其来源运营商，或通过PING来判断其访问哪条链路速度最优，然后通过VIEW功能将其解析到不同的虚拟服务器上。

和大多数网站的通行做法一样，我们使用了Nginx来分离静态资源，使用Memcached来做数据级缓存。页面级缓存和数据级缓存二个缓存层的应用，使得天涯数据库的直接访问不断下降，以往数据库高负载、响应恶化而导致网站显示延迟等问题不断减少。数据库越来越成为数据存储容器而不是数据处理机器。

天涯网络和应用架构在研发团队及运维团队的共同努力下已基本成熟稳定，天涯每天的PV从千万级支持到亿级。

阶段三 统一的运维管理平台

在天涯运维的早期阶段，运维工程师直接采用开源软件，以及自行开发运维监控工具，没有统一的平台来管理，例如我们用Cacti来做性能监控，Nagios做主机监控，PHP+SSH实现远程管理，ASP+MSSQL写资产管理系统，PHP+RRDTool实现LVS管理等，各个运维工具开发语言不同、各自为政、信息孤立、无法打通共享，也不利于统一的维护。

2009年，我们开始组建独立的运维开发组，负责天涯运维管理平台的规划和开发。将运维工具和资源统一起来，开发平台化的天涯运维系统。

我们以资源管理中心为核心，管理服务器、网络设备、域名、产品、机柜、IP、VIP等信息。各个工具和平台通过资源中心进行数据分享和关联。我们不再使用Cacti来监测性能，改为自己开发的性能监控系统。每台服务器上统一部署性能代理，通过执行插件的方式进行数据采集。例如基础插件可以获取主机的CPU、内存、磁盘、网卡等基本性能。扩展插件可以采集指定服务或进程的状态数据，支持各种开发语言，如Shell、Perl、Python和C等。

无论是基础架构运维还是产品运维的目标异同，我们最终的目标是给用户提供高品质的网站服务。因此运维和研发团队的协作和沟通应该更紧密、更透明，以能给用户带来更好的用户体验 。

例如要监测Varnish的运行状态，我们用Python编写varnishStats插件，插件通过Varnish下的varnishstat命令获取client_conn、cache_hit和cache_miss等Varnish状态数据。插件定期在目标服务器上执行并取得数据后，通过消息回传到消息服务器，由消息服务器根据插件预定义规则进行处理。在报警规则定义上，工程师可以按产品或服务器的优先级分别定义短信、邮件等报警方式。我们采用了串口的短信猫来发送报警短信。

系统定期从后台按产品、分类等汇总性能，通过报表能直观掌握几十个产品的运行状态，便于各个产品团队针对目前的产品运行现状决定产品的优化和扩容方向。

消息服务器可水平扩展，支持分布式机房，因此易于扩展，能支持更大规模的服务器集群环境。处理的数据支持关系数据库和环状数据库，可根据处理对象的不同而采取不同的数据处理和存储方式。

在开源Func的基础上二次开发了远程操作管理。Func是由红帽公司以Fedora平台统一网络控制器，用来解决统一管理监控问题而设计开发的系统管理基础框架。运维工程师可使用Shell或Python编写和添加各类操作模块，并按服务器的产品属性授权给不同角色的工程师来批量远程操作，提升了服务器管理的工作效率。

我们使用开源的Puppet作为中心化配置管理系统，轻松管理数据中心数千台规模的服务器，管理内容包括文件、配置和用户等资源。我们将Puppet的操作嵌入运维平台中，采用SVN同步，这样可以进行权限控制，将Puppet上的配置变更纳入变更流程里。

已完成的天涯运维系统模块还包括数据库信息管理、网络设备管理、容量报表、应用发布、LVS管理、虚拟机管理、HAProxy管理等功能模块。正在开发的模块有软件包管理、应用和业务监测、流程管理、工单管理、数据库管理等功能模块。

公司的业务模式及规模差异决定了运维团队的工作方向。因此，我们在借鉴各互联网公司运维经验的基础上，结合天涯自身的业务和产品特点，开发适合天涯运维的运维管理平台。

虚拟化

我们从2009年开始探索使用虚拟化技术。在虚拟化前，我们受到一些问题的困扰，例如主机性能利用不均衡、产品分散而造成主机闲置浪费，快速增长的产品和滞后的硬件采购部署周期的矛盾等。我们期望通过虚拟化来解决上述问题。

经过测试和实践后，我们选择了Xen来实施虚拟化（后期因技术策略的调整转向KVM），使用性价比较好的服务器硬件方案来部署虚拟化，配置为双路4核CPU、64GB内存、6块SATA 7200转硬盘和RAID 5容错。普通虚拟机的实例为单核CPU、8GB内存、100GB磁盘空间和100M网卡。实

际使用中我们会根据性能负载和虚拟机的业务特点来动态来调整虚拟资源。一些虚拟机甚至动态调整到8个虚拟CPU或更大的内存。我们会根据物理服务器和虚拟机两者的实际负载来决定其上虚拟机的资源分配。我们没有使用统一存储，因为统一存储性价比不高，不适合低成本运作的互联网公司。我们将同一产品的虚拟机部署在不同的物理服务器上来降低硬件故障的风险。应用集群通过Keepalived+LVS和Keepalived+Haproxy两种负载均衡方案来解决。

在初始化物理机时，我们通过管理工具和模板在物理机上自动生成固定数量的虚拟机，业务需要时人工唤醒这些虚拟机，通过自动化管理工具来自动部署软件包、应用包和配置，将其加入到应用集群中。这样可以避免线上创建虚拟机时消耗大量的I/O操作，从而影响其他的虚拟机性能。

天涯虚拟化平台的选型详见蒋清野的文章《开源IaaS软件的比较——构架、功能、社区、商业及其他》（网址http://www.qyjohn.net/?p=1624）。天涯在Convirt开源社区版的基础上进行了二次开发，以满足天涯业务模式的需求。

经过三年的虚拟化实践，虚拟化技术很好地解决了天涯对各产品的线上扩容和缩容，亦满足了低成本和运维人力成本，实现了运维团队对业务的快速支持和响应。

虚拟化在天涯应用以来较大的问题是虚拟机的资源争抢，一些资源消耗大的虚拟机对本物理机上的其他虚拟机会造成性能影响，从而影响其他业务的稳定性。我们的策略是对物理机和虚拟机的总体性能做综合的监测，跟踪某些资源消耗快速增长的业务，针对这些业务做个性化的处理，例如快速扩容，或者独立采用特定的物理机资源池来负载特殊应用的虚拟机。

另外，对一些不适合虚拟化的业务如高磁盘I/O、密集计算、高并发大流量等的应用，我们仍然根据业务特点使用物理机来承载。

随着天涯虚拟化的不断深入，天涯的虚拟化应用越来越广泛，目前天涯的虚拟化已占了天涯服务器总数的40%以上，这个数字还在不断扩大。

结束语

从天涯运维发展的道路来看，我们仍有太多不足的地方，天涯产品的健壮性也达不到业界的标准。

首先，运维团队过分注重基础架构的可用性，较少关注产品的可用性，对产品运维的支持力度不够。其实无论是基础架构运维还是产品运维的目标异同，我们最终的目标是给用户提供高品质的网站服务。因此，运维和研发团队的协作和沟通应该更紧密、更透明，以能给用户带来更好的用户体验 。

其次，在运维平台的开发上我们过多地站在运维角度来思考，没有从业务角度去规划运维系统。运维系统过于偏重基础架构。我们会在下一个阶段来解决这个偏差。

最后，最困扰我们的是人才问题。天涯的运维团队在海口这个三线城市，既没有一流的高校也没有互联网产业氛围。工程师招聘只能通过校招，人员经验少，流动大，给运维工作带来很大的挑战。我们只有通过成熟的运维平台、流程规范的不断完善和建立学习型团队来弥补这方面的弱势。

希望我们天涯的网站运维历史能给大家带来一些可借鉴的经验和思路。

周小军

2002年以来就职于天涯社区网站，任天涯社区技术中心云计算与基础架构部副总监，负责网站基础架构运维等相关工作，具有大中型互联网站的运维及架构经验。

通往部署自由之路

文 / 常新居士

新挑战

互联网的兴起与技术进步似乎正将我们推到能力的边缘。如今，一款新产品从需求到用户，其间隔越来越短——以年为单位的发布传统已成为昨夜星辰；而同时，新的方法论和实践强调软件应及早、频繁地构建与交付。此种情势下，软件产品的部署工作在软件生命周期中承担着愈发重要的责任。

对于简单的软件产品，相关人员的手工部署就已足够应对，例如多数互联网产品的初创期，为了快速占领市场，常诉诸于“人肉糙快猛”。当软件规模变大、发布更为频繁时，企业便开始着手建立专职的运维团队，然而手工部署往往并不能满足及时、高效和正确部署的要求，因此，以持续集成为基础的自动化部署——配合Puppet这样的配置管理系统，成为多数企业采用的行之有效的方案。由于该方案的出色表现，许多拥有若干产品的大型企业也开始采用它来构建自己的部署系统，但这样的系统是否可以达到今日产品和业务所要求的灵活、快速、一致和高扩展的要求，并在企业内部有效消除不必要的工作，最终使运维工作与成本大幅降低呢？答案也许并非你所想的那样简单……

再看部署

对于部署，Ezra Zygmuntowicz在《Deploying Rails Applications》中给出了一个简单比喻如图1所示。在Ezra看来，部署——其最简单的形式——就像是搬家：找到房子并将打包好的东西搬进去，当然，少不得布置和打扫的工作。由此看来，部署工作应该是所有计算机使用者参与最多

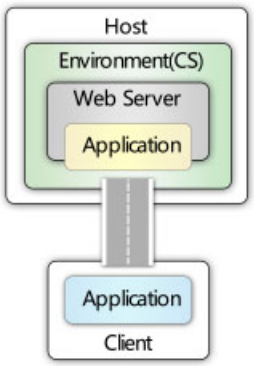


图1　简单化的部署系统

的活动——因为任何软件的安装其实都是该软件的一次部署。

那么对于软件产品，其开发和生产环境又是如何构成的呢？除了安装项目的软件包，我们还需要重点处理四类问题：依赖、运行环境、配置信息和部署环境。

依赖

现代软件开发不可避免地要依赖其他系统或框架所提供的功能或代码。而这些系统或框架也各自对其他系统或框架有所依赖。由此，系统间的依赖关系会形成一颗依赖树。根据项目的生命周期，这样的依赖可能有所不同，例如编译时依赖、测试时依赖及运行时依赖。需要注意的是，虽然编译时依赖常常是运行时依赖，但并不能推断出一方必然是另一方。例如在开发的过程中需要某些提供API的Jar包，而运行时可能是具体API实现的Jar包。当项目部署到生产环境时，所有的运行时依赖必须就位。

运行环境

项目除了对程序包的依赖，对于运行环境也有些具体的要求，例如Web应用需要安装和配置Web

图2 统一的部署概念和操作

服务器、应用服务器、数据服务器等，企业应用中可能需要消息队列、缓存、定时作业，或是对其他系统以Web Service方式暴露的服务。这些可以看做项目在系统层面对外部的依赖。这些依赖有些可以由项目自行处理，有些则是项目无法处理的，比如运行容器、操作系统等，这些是项目的运行环境。

配置信息

当项目部署到目标环境时，都要进行相应的配置工作。这些配置信息或作用项目所依赖的环境，或作用于项目本身。当一台机器上部署多个项目时，彼此的配置信息应该互不影响。

部署环境

项目会依其使用目的的不同被部署到不同的机器中，例如生产服务器、UAT服务器、开发机。而项目在不同环境中其行为可能会略有差异，因此，项目必须可以了解其部署环境的信息。

对于简单的项目，我们完全可以手工安装运行环境，将所依赖的包打包到我们的项目中，并通过脚本来进行环境的配置工作。当然，如果使用Puppet的话，部署的工作会更为便捷。但当项目变的复杂时，尤其是同时运行维护着多个项目和产品线的企业中，部署工作将遇到怎样的挑战?

复杂的部署

先看项目的变化。通常，项目代码被集中在一处，其外部依赖都是第三方软件。当企业中项目变多，或项目本身的业务变得复杂时，项目将会被拆分成若干的子项目，以便于解耦（如业务代码与配置脚本放置在各自的包中）或者重用（如抽取出公共模块以便其他项目使用）。这时，部署的工作不仅要将分散的代码组合起来，而且要解决项目和子项目的依赖关系，而软件间的版本依赖使这个问题变得更为复杂。

与此同时，项目不仅要部署到不同的部署环境中，而且可能会牵涉多个业务区域。当部署涉及的机器增多时，目标环境的操作系统也将影响项目的构建和部署。

在这样复杂的项目环境中，有哪些常见的问题呢?

■ 场景1：环境升级

项目A和项目B都依赖于Web容器，公司决定升级Web容器版本，而公司要升级的机器有上百台，依赖人肉升级已不现实，维护团队因此针对各种软件开发了相应的自动化脚本，但当新的软件出现时，必须要开发新的脚本。而且当同时升级若干环境软件时，则难度随之增大，手工调度的方式极易出错，当升级失败时仍需要大量人工处理。由于存在大量升级脚本，有一定的维护成本。

■ 场景2：升级与回滚

针对环境升级，公司为项目A和项目B开发了新的版本。但环境的升级和软件的升级不是同步进行，出错的可能性非常大（想一想间接依赖和多重依赖的情况）。当新版本部署到生产系统时，发现问题需要回滚到之前的版本——所有运行时版本都需要回滚，而且环境也需要同步回滚。

■ 场景3：运行时依赖

在简单项目中，我们可以将所有的运行时依赖都打包到一起。当项目依赖关系复杂时，这样产生的包将非常臃肿，潜在地延长了部署的时间（想一想全世有几百台服务器，一个部署计划需要部署几百兆文件的情况），而且产生冲突的可能性非常大，加之对于不同类型的项目（Java和Ruby项目）缺乏通用性。2006年左右，某电信企业是在项目的开始，通过Excel来统计运行时依赖，牵涉若干项目组，反复多次，耗时耗力。

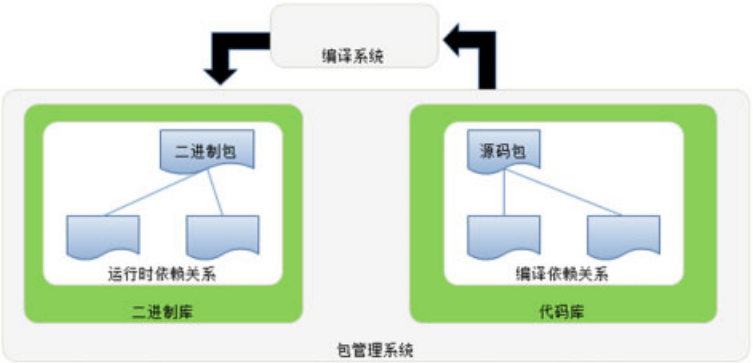


图3 抽象的包管理系统

■ 场景4：不一致的环境

简单项目中，开发环境和运行环境都由开发人员搭建，当公司变大时，系统的运行环境将由运维人员搭建，而开发环境如果由运维人员搭建则工作量太大，由开发人员自己搭建则操作复杂又容易产生不一致的情况。而且多种部署环境，以及每种部署环境中大量的机器，加剧了产生不一致的可能性。

■ 场景5：热切换

对于某些部署，需要尽量减少服务的停止时间，需要在服务的同时进行部署。

■ 场景6：业务扩展

当业务扩展到新的区域时，维护人员要从事大量重复性的工作。

大型企业人多、项目多、机器多、项目环境复杂、部署维护工作繁多。企业要应对复杂的项目环境和各种不同的部署要求，而且以项目为中心开展部署工作时，企业内又会产生大量重复性的投入。究其根本，大型企业中的部署不再是一个简单的问题，而是一个部署生态圈，基础设施和环境管理必须要纳入考虑之中。要实现真正意义上的自动部署，我们就必须把环境和项目同等对待，统统纳入管理之中。同时，部署本身要得到统一。**一个好的部署系统，应该是易于建立、易于使用、易于维护。**对于使用者来说，可以做到一键式部署。

部署系统的建立

部署系统的建立，重点要解决三个问题：包管理、环境管理、部署管理。

包管理

如果读者接触过早期的Linux，那么软件的安装一定给你留下过深刻的印象——安装的过程中，会不断发现依赖的包缺失。

随着Linux的软件包管理逐渐成熟，现在，我们越来越多地依赖Yum、apt-get等工具来进行软件安装。甚至连编程语言也开始建立自己的包管理器，如Ruby的RubyGem、Node.js的npm等。

部署系统必须对软件的包进行管理，流行的包管理机制不乏借鉴之处，例如软件包的命名、类型、优先级、状态和依赖。这里值得一提的是，某个项目包在部署系统中可能存在着两种类型的软件包：源代码包和二进制包。源代码包通常会放置在SVN或Git这样的版本库中，相应的，二进制包也必须进行版本控制，并统一存储；两种版本之间的关系需要保存下来，以便生成对应的依赖树。另一个需要考虑的问题是，为了方便未来的部署工作，二进制包的结构必须进行规格化。

环境管理

什么是环境？系统运行所依赖和包含的一切就是其环境：硬件、操作系统、网络资源（IP地址、域名）、服务容器等。对于部署而言，广义上，这些统统应该纳入环境管理的范畴，但狭义上，从软件系统的角度看，一个环境就是其运行需要的软件及其配置（我们先把操作系统和网络资源当做基础设施，其在部署时已处于就位的情况）。

项目的生产环境 = 项目本身的软件包 + 项目运行时依赖的软件包 + 项目运行时依赖的其他软件 + 项目的配置信息

由于，项目本身的软件包、项目运行时依赖的软件包，以及项目运行时依赖的其他软件在本质上没有区别——都是软件，上面的定义可以进一步抽象为：

环境 = 软件包 + 配置信息

在这个定义下，我们必须改造项目所依赖的软件——以包的形式导入到公司的整个项目资源库中，比如Apache将作为一个包被导入，而Apache

依赖的其他包也将依次被导入，并建立起正确的依赖关系。同时，在导入的过程中还必须做些相应的调整，例如环境变量的读取和设置，必须来自于环境配置模块，而不要修改系统的环境变量，防止不同项目在系统环境配置上相互影响和依赖。

再回头审视我们的示例，项目的生产环境可以部署在不同的区域，对于各个区域可能有定制化的设定。这就像面向对象中的类，可以通过继承使子类重用父类的公有属性和行为并添加自己特有的信息。

通过这样的关系，我们很容易为业务扩展建立一种简单的结构，由于在根环境上定义了软件包和配置项，派生环境很容易做到一致的部署。环境依然是处于知识层面（Knowledge Level），它并未与具体的基础设施相关联。

当我们将一个环境“具现化”成一个运行系统时，我们就产生了一个真正的环境实例。在这两者之间，我们还必须要考虑环境实例的使用目的及安装所依赖的其它信息（如机器）。因此，我们需要增加一个环境目标来集中这些信息，而且由于不同目标的环境可能会有所差别，因此，环境目标也需要配置的能力。

环境实例是如何产生的呢？部署，一次部署可能会产生一个环境实例。一系列部署将产生对应于环境目标的多个环境实例，除去当前起作用的环境实例外（最新的），其他的是历史环境实例。通过在历史环境实例中切换，我们自然而然就可以实现整个环境回滚，因为项目所依赖的一切都已成为环境中的软件包，而且环境依赖的包的版本会随着部署具体确定下来。如此一来，我们便可以给每个环境实例分配一个版本号，再通过环境实例的版本号与软件包的版本对应起来，从而得知一次部署时应用的具体软件包。

目前的设计，已经具有满足一致的部署、业务水平扩展、性能水平扩展、以及环境升级和部署回滚的能力，并且可以根据使用目的的不同部署对应的环境（开发、测试、生产）。

接下来，让我们看看部署在这样的设计下是如何工作的。

部署管理

根据我们的设计，当建立的目标环境后，对目标环境的部署将生成具体的环境实例。回顾我们在文章开头给出的比喻，部署就是个找房子搬家的过程，既然是过程，自然涉及到一本步骤，以及各个步骤中要执行的活动。对于标准的部署——安装软件包并启动环境，可能的步骤将会是：选择将要部署的软件包的版本；生成新的环境实例（确定环境实例的版本和其依赖包的版本，确定环境配置等）；清理和准备目标机环境……

而对于停机或重启之类的部署，可能的活动将大大简化。结合这些信息，我们发现，部署其实质就是，按照其目（部署类型）的执行一系列步骤将环境置于其目的所指向的状态中。由此，我们将部署过程抽象为如下结构，并将其与环境管理结合在一起：

包管理、环境管理和部署管理各就各位，我们可以将各个项目环境纳入我们的环境管理之中，甚至是开发工具本身。由此，整个组织在部署上的认识和操作将保持一致。

下一步

这次探讨的主题到此结束，然而，对于构建企业的部署系统来说，这仅仅是个开始。尚有诸多的问题我们还没有展开讨论，例如：操作系统对部署系统的影响，可扩展、灵活的配置系统，环境管理如何支持开发调试等。

路漫漫其修远兮，吾将上下而求索。我相信大家在实现部署系统时，将会结合企业的业务特点找到自己的答案。

常新居士

混迹于大型软件企业多年，从事架构、设计与编程语言方面的工作。目前负责公司数据挖掘与预测。曾合译《管理3.0》、《Designing Interfaces》等书。

集中化运维管理

Puppet管理之路

文 / 刘宇

大数据时代高伸缩性、容错性的特点给运维提出了更高的要求。系统管理不再是疲于安装操作系统、对系统参数进行逐一配置与优化、打补丁、安装软件、配置软件、添加某个服务的时代。为了提高效率、避免重复劳动、减少错误、积累知识，系统管理员都已开始做一些局部的自动化工作。但这些还远不够，为了满足运维需求，需要更彻底地应用自动化运维工具。

本文将介绍如何利用配置管理自动化工具Puppet完成系统安装、监控报警工作，解剖Puppet给系统管理员带来的便利，同时还将介绍Puppet的架构和工作原理。

从系统安装到自动化部署软件、配置、回滚，再到服务器的可用性、性能、安全维护，运维管理人员都需要完全掌握，为了有效完成工作，熟悉几款优秀的开源软件必不可少。如表1所示。

表1 常用运维工具分类

系统安装	配置管理	监控报警
KickStart	Cfengine	Nagios
Cobbler	Puppet	Zabbix
	Capistrano	Ganglia
	Func	Cacti

对于我来说，工具箱中最趁手的要数Kickstart、Puppet、Zabbix和Cacti。

运维工作难点

运维工作流程

常见的运维工作流程包括：安装系统→优化系统与配置→安装软件→配置软件→添加监控→检查。后续可能还会有添加服务→配置变更→打补丁修复漏洞等。是不是觉得很繁琐？尤其当你负责大量设备，无法凭借一己之力完成时，便需要一些工具来帮忙。

运维工作面临的各种不确定性更让人头疼。在10台机器上变更应用还是很简单的事，但如果上升到千台、万台就会变得非常复杂。重复性的劳动还会让人觉得疲惫和乏味，久而久之可能还会产生厌倦工作的情绪。使用Puppet则能将这些问题迎刃而解。

自己实现自动化

为了提升工作效率，减少出错机率。很多公司都在逐步采用自动化来实现上述工作。有的公司选择自己开发一套工具，因为可以按照需求任意定制，但这真的有必要吗？我们来看看这样做的缺点。

1. 从头造轮子：构建脚本工作的挑战与繁琐。
2. 程序的可维护性无法保障（语言）。
3. 支撑不同的平台。
4. 系统重装后的考虑。

统筹与规划整个系统需要花费很长时间，伴随着人员的流动，技能水平不一，还会带来新的问题。而且单独开发的系统不可能只是支撑一个平台——跨平台开发则意味着更多不确定性。

自动化配置工具比较

表2比较了两种最常用的自动化运维工具Puppet和Cfengine。

表2 Puppet和Cfengine功能对比

	Puppet	Cfengine
使用用户	Google，Red Hat	Twitter
开发支持	Foreman	
商业运行	Enterprise	Enterprise
程序语言	PHP，Django，Ruby	Ruby
文档环境	Mail-Lis，IRC	Mail-List，IRC
平台支持	All OS	相对少一些
现成实例	Example42	
依赖关系	灵活处理，图表输出	

但我真正想说的是：以上比较没有太多的意义，工具在于你怎么去用，如何用得最好，发挥出它的优势，与你的业务完美结合。我们不需要忙着选工具，而是应该深入研究它。

剖析Puppet

在使用任何软件前我们都需要了解其工作原理，否则会给后续使用带来诸多不便。Puppet采用了非常简单的C/S架构，所有数据的交互都通过SSL进行，以保证安全。它的工作流程如图1所示。

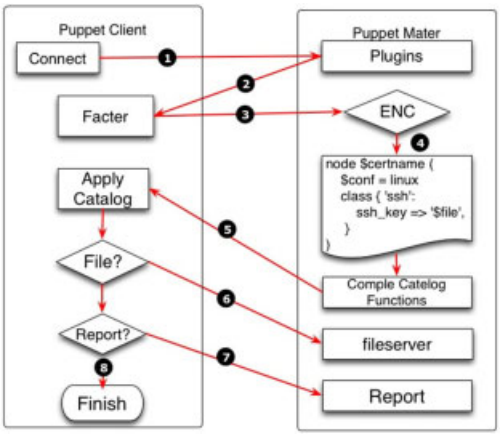


图1 Puppet工作流程

1. 客户端Puppetd向Master发起认证请求，或使用带签名的证书。
2. Master告诉Client你是合法的。
3. 客户端Puppetd调用Facter，Facter探测出主机的一些变量，例如主机名、内存大小、IP地址等。Puppetd将这些信息通过SSL连接发送到服务器端。
4. 服务器端的Puppet Master检测客户端的主机名，然后找到manifest对应的node配置，并对该部分内容进行解析。Facter送过来的信息可以作为变量处理，node牵涉到的代码才解析，其他没牵涉的代码不解析。解析分为几个阶段，首先是语法检查，如果语法错误就报错；如果语法没错，就继续解析，解析的结果生成一个中间的“伪代码”（catelog），然后把伪代码发给客户端。
5. 客户端接收到“伪代码”，并且执行。
6. 客户端在执行时判断有没有File文件，如果有，则向fileserver发起请求。
7. 客户端判断有没有配置Report，如果已配置，则把执行结果发送给服务器。
8. 服务器端把客户端的执行结果写入日志，并发送给报告系统。

当服务器超过千台

当你的服务器越来越多时，你可能会发现Puppet执行效率开始下降，服务器已无法满足你的需求。下面介绍几种方案来解决这类问题。

LoadBlancer

这是通过非常简单的扩容Master方案，来提升Master计算“伪代码”的能力。通常这种架构能支撑1000台左右的服务器。当然，这也依赖于你的系统是否够“复杂”。

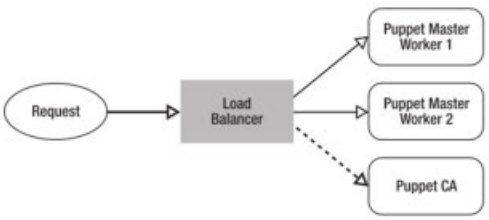


图2 LoadBlancer方案

这种架构有两种常用的实现方式：Apache+Passenger，以及Nginx+Mongrel。本文将以后者为例简单介绍其工作方式。

1. Puppet Master运行多个进程：

Puppet Master+Mongrel，port 18140
Puppet Master+Mongrel，port 18141
Puppet Master+Mongrel，port 18142
Puppet Master+Mongrel，port 18143

2. Nginx通过Upstream的方式实现对Puppet Master的负载均衡。Nginx监听port 8140将除文件下发之外的请求，代理转发给上面的4个Puppet Master实例之一，Nginx会对客户端证书进行验证，但需要配置CA签发的证书才允许请求，我们也可以再配置8141提供证书签发。

3. 如果使用了fileserver，Nginx也可以直接处理。

Puppet认证负载均衡

仅有多个Master是否够用？一台机器还是有风险，为此我们需要有容错，将Master分布在不同

机器上，而CA认证也是非常重点的一部分，我们可以采用以下架构做一个热备。如图3所示。

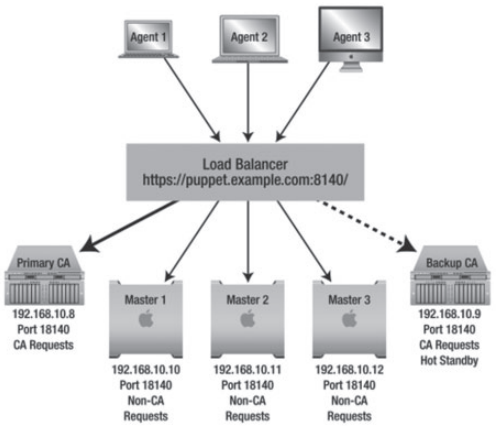


图3 Puppet认证负载均衡方案

这架构还可以进行扩展。我们再次回顾Puppet工作原理；Puppet客户端跟Nginx之间是HTTPS连接，Nginx跟各个Mongrel之间用的是HTTP连接。对客户端证书的验证由Nginx负责，而Nginx只需要有CA的公钥就可以做验证工作。这样做的好处是多台管理机之间不需要同步客户端的证书等设置，只需要有CA的公钥，公钥复制就能使用。但这样有一个缺点：不太方便删除客户端证书。不过可以采用一个主管理机的方式，其他管理机从这台管理机上实时同步证书。

Puppet管理机集群思路如下：

1. 将CA配置同步到每台机器上，包括公私钥；
2. 用CA给每台管理机器签发一个证书；
3. 每台管理机都配成LoadBalancer的方式，8140提供配置管理，8141提供证书签发；
4. 各管理机之间可以使用Keeplived实现高可用性及故障切换，包括HA等，架构可随意扩展；
5. 每台管理机配置分Production和Development两种，简单地通过Git中发布到管理机上；
6. 测试时只修改Development部分，在个别客户端指定用它，成功后再推到Production下；
7. 配置一台主CA管理机，解决删除认证的问题。

合理规划

所有的一切事后挽救方案都不如使用前合理规划，你需要非常清楚当前业务的状态、运维的现状。了解你需要解决什么问题，然后将它分解，再逐步击破。以下几点需要注意：

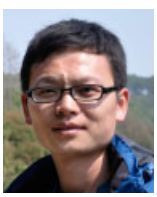
- 推荐采用Git管理Puppet；
- 规范HostName，采用DNS管理；
- fileServer独立，将不经常变化的放在fileserver里，经常变化的放在模板中；
- 沟通自定义OS。

可能很多人不太明白为什么要自定义OS，它最大的优点就是可以在系统初始化安装时帮你做一些Puppet所需要的软件包，通过采购设备时拿到的SN号，在WebUI系统里注册这台机器的信息，机器在启动后即可完成所有的配置。如果你的WebUI做得更好，可以调用监控系统的API完成监控。这样是不是很完美？

结束语

相信大家读完本文后不但对Puppet有了整体的了解，而且会更加熟悉自动化运维工作的重点。也许会让你开始考虑在自己的运维工作中，尝试采用Puppet解决诸多重复性劳动，或者解决你现在所面临的架构问题。

我想对很多希望学习Puppet或正在使用Puppet的系统管理员说，工作原理很重要，很多人就是没有弄明白工作原理，因而在使用过程中一遇到问题就手忙脚乱。读者朋友们一定要靠多动脑思考来解决问题。



刘宇

linuxtone.org创始人之一，SinaEdge平台运维主管。负责新浪微博、新浪视频、看点、微盘、音乐等业务CDN运维。曾编写《Puppet集中化管理》。

创业团队服务器运维工具集

米聊服务器端的开源选择

文 / 陈臻

2010年秋天，我加入了当时还名不见经传的“小米工作室”，此后的两年，我见证了移动互联网异军突起、群雄逐鹿的开荒时代，感慨无限。特以此文记录如何在一个集齐各大公司工作背景、技术方向五花八门的团队环境中，快速选择和定位服务器端所使用的开发与运维技术方向。本文只介绍介于开发与运维之间的一些便利的、真正可以在创业团队中用得上的工具，文中的一些观点仅供参考，具体问题还需具体分析。

表1　本文提及的工具及其功能

本文提及的工具	功能
Resin	Java容器首选
Rose	框架开发统一思想
Thrift/ZooKeeper	高可用服务内部通信必备
Scribe/Hadoop/Hive	日志收集和统计

Resin

Java为主的技术团队，一定需要选择一个Web容器来运行自己的业务代码，可能许多人把Tomcat作为唯一选择，但实际上Resin作为一个商业级的服务器应用，已在各大公司的Java团队中崭露头角。Resin凭借其多样化的功能、完整的watchdog机制、快速简单实现负载均衡等特点，在许多创业团队中备受青睐。

在使用Resin时，它的主程序会启动一个叫watchdog的进程，顾名思义，这个进程主要负责了所有Resin进程的状态监测。一般情况下，可以在一个Resin服务器中配置多个实例，运行时将会是独立的JVM进程，以及一个独立的watchdog JVM进程。每个Resin实例都需要配置一个watchdog-port来支持watchdog的检测，从而做到当一个进程意外终止时，watchdog都能将它重启。Resin 4支持在resin.xml中设置JVM的参数，有一些设置经验可供参考。

- -Xms与-Xmx的值最好相同，避免每次垃圾回收完成后JVM重新分配内存。-Xmn的值大致设置为-Xmx的1/4～1/2即可，此值对系统性能影响较大，Sun官方推荐配置为整个堆的3/8。
- 如果Spring应用出现了大量动态Class挤爆PermGen Space的情况，Resin会出现500错误，可以考虑通过增大“-XX：PermSize”和“-XX：MaxPermSize”这两个参数来避免。

有网友用Nginx 1.2.0与Resin4.0.29进行压力测试同一个静态文件，也得到了它们吞吐能力旗鼓相当的结论。

Rose框架

选择它是因为这个框架可以很快上手，学习成本非常低。一个有其他类Java语言经验的工程师，也可以放心大胆地下手写业务代码。同时，即便是从未接触过它的工程师也可以在两天内完全掌握这个框架的各种特点。

其HTTP路径内部以Tree的形式保存，天然支持RESTful格式的URL，使一个项目会得到非常标准的路径，为未来进行各种灰度升级或权限判断提供了很不错的基础。

一个使用Rose的简单Controller例子如下：

```
@Path("hello")

public class HelloController {
    @Get("world")
    public String getWorld() {
        return "topiclist";
    }
}
```

部署完成后，可以通过http://localhost/hello/world访问以上代码，一个完全没用过Spring的人在了解这一规则后就能开始修改代码了。

Rose框架的DAO语法完全以Interface方式剥离，SQL汇于一个类中，方便DBA进行Review。

一个简单的DAO例子如下：

```
@DAO
public interface TestDAO{
    @SQL("insert into test (id,msg) values (:t.id,:t.msg)")
    public void insertTest(@SQLParam("t") Test test);
}
```

例子中传入了一个Test对象，在SQL的注解中可以用简单的语法进行解析，生成最终的SQL。这在实际项目中非常方便。利用Spring对context的管理，可以简单配置连接池等提速环境。

此外，Rose还可以多线程并行完成多个业务过程。除性能优势外，可以在完全分模块开发后，再用这个并行特性进行组合，从而做到“多条线同时开发，最终合并得到结果”的效果。

```
@Get("/3.8")

public String pipe(Pipe pipe) {

    pipe.addWindow("p1", "/wp1");
    pipe.addWindow("p2", "/wp2");

    return "pipe";

}
```

同前面的Controller的代码一样，当访问/3.8时，Pipe会在所有的Controller中寻找其需要的逻辑，以上例子中所提供的两个路径分别是/wp1与/wp2，这两个路径对应的逻辑又可以被并行执行。在团队作战时，/wp1与/wp2分别交给不相关的人去实现，最终进行组合即可，这样非常方便。

同时，Rose还有诸如自定义拦截器、参数解析器、异常统一处理等大量细节功能，都是对使用Spring的最佳实践总结。对这一内容感兴趣的读者，可以关注Rose的快速入门文档：http://www.54chen.com/rose.html。

Thrift/ZooKeeper

许多人认为作为内网通信框架，Thrift没有完善的管理体系。但经过长时间的实践，我们使用的Thrift+ZooKeeper的方式十分稳定。它现在能够依赖ZooKeeper做到高可用，依赖Thrift的优秀设计做到高性能。从SOA方面考虑，创业公司选择这个组合非常明智。

Thrift的定义文件对版本控制做得非常好，比起自己用PB来做要方便很多。下面是一个简单的Thrift文件定义的例子，目的是完成一个名为Hello的服务，其中只有一个方法，当这个方法被调用时，执行hello方法的实现，Thrift格式代码如下：

```
#!/usr/local/bin/thrift --gen java

namespace java com.chen

service Hello{
i32 hello(1:i32 x,2:string world)
    }
```

这个文件是Thrift自己定义的格式，依靠数字序号来做版本控制。在版本与兼容方面，有一些大家都需要遵循的最佳实践。

- 不修改已存在的序号，因为在服务集群中间可能会存在不同版本的数据交换，如果修改了已存在的序号，那说明协议发生了根本性变化。
- 任何新增字段都应该是optional的，这是Thrift为了做上下兼容专门设置的一个关键词，在定义参数时可以使用。
- 永不删除序号，道理同第一条。
- 尽量不修改默认值，尽量做到定义的数据在任何时间都保持其唯一语意，不会在未来带来各种不兼容的麻烦。

ZooKeeper的高可用集群、快速感知机制，使其正在成为大规模高可用服务集群所必备的工具。但其官方客户端要需要进行大量封装。我推荐来自Netflix公司的开源Curator客户端，它的实现比较优雅，语法基本不需要学习，且有大量的组件可选择，用以配合ZooKeeper达到专业的业务效果。

下面是一段使用Curator的例子，操作位于zookeeper.n.miliao.com服务器2181端口的ZooKeeper服务，并对这个ZooKeeper的路径进行了创建和删除操作（代码见下页的代码1）。

代码1基本包括了所有的ZooKeeper操作，ZooKeeper与Thrift的搭配，主要用于线上自动感

知Thrift节点的动作，如果有节点加入或者退出，客户端调用者可以动态地进行处理，以实现高可用性。

```
public static void main(String[] args) throws Exception {

    String path = "/test_path";

    CuratorFramework client = CuratorFrameworkFactory.builder()
        .connectString("zookeeper.n.miliao.com:2181")
        .namespace("/brokers")
        .retryPolicy(new RetryNTimes(Integer.MAX_VALUE, 1000))
        .connectionTimeoutMs(5000).build();

    // 启动 上面的namespace会作为一个最根的节点在使用时自动创建
    client.start();

    // 创建一个节点
    client.create().forPath("/head", new byte[0]);

    // 异步地删除一个节点
    client.delete().inBackground().forPath("/head");

    // 创建一个临时节点
    client.create()
        .withMode(CreateMode.EPHEMERAL_SEQUENTIAL)
        .forPath("/head/child", new byte[0]);

    // 取数据
    client.getData().watched().inBackground()
        .forPath("/test");

    // 检查路径是否存在
    client.checkExists().forPath(path);

    // 异步删除
    client.delete().inBackground().forPath("/head");

    // 注册观察者，当节点变动时触发
    client.getData().usingWatcher(new Watcher() {
            @Override
            public void process(WatchedEvent event) {
                System.out.println("node is changed");
            }
        }).inBackground().forPath("/test");

    // 结束使用
    client.close();

}
```

代码1

Scribe、Hadoop、Hive

Scribe、Hadoop和Hive是后台统计分析三样宝，有了这三样工具，任何用户行为都可以被记录下来用作分析，为各种决策提供依据。

Thrift升级快，Scribe升级慢，我们试过的最好的搭配版本是在0.5版Thrift下编译的Scribe。此处不再讲述如何编译安装，而是探讨实践过程容易遇到的问题。

要实现在代码里插入Scribe打点的Client代码，最可靠的方式要属实现一个log4j的异步appender，具体实现办法网上有很多，此处不再赘述。

Hadoop的节点在长时间运行后，数据越来越多，任务也越来越密集，当发现有任务失败时，可能会在datanode上发现一些日志，例如像下面这种：

org.apache.hadoop.hdfs.server.datanode.DataNode: writeBlock blk_8964076545845199727_216399 received exception org.apache.hadoop.hdfs.server.datanode.BlockAlreadyExistsException:

该情况说明datanode在写入时遇到了服务不响应的问题，如果不是内网环境故障，很有必要检查一下当时的负载和各个节点的环境配置。特别需要提醒的是，新的节点是不是设置了ulimit的默认值，可以通过ulimit -SHn 18912来查看。

Hive在长时间运行后，可能需要进行迁移，这时会有各种环境问题冒出来。一般来说，如果要迁移namenode，依照以下步骤操作即可高枕无忧：

- 复制打包老的namenode；
- 复制打包老的namenode的HDFS目录；
- 到新机器准备好：修改master的值，复制到各节点，修改Hive的定义；

因为这些信息会被写死在Hive的DBS和SBS表里，所以Hive使用了MySQL来保存metadata，接下来是关键的一步。进入MySQL，执行下面的SQL语句：

update DBS set DB_LOCATION_URI=REPLACE(DB_LOCATION_URI,'old host','new host');

update SDS set LOCATION=REPLACE(LOCATION,'old host','new host');

再启动后，迁移Hive namenode成功！

结语

创业公司没有专门的运维团队，大多是开发人员承担，一开始的选择往往会对未来造成深远的影响。大多数时候，选择哪种工具需要有一些原则，我们一直秉承的原则是：1. 团队中有人通读过代码且能搞定；2. 选择大公司正在使用的工具。这两个原则满足其一即可。

本文内容来自我们在开发运维中遇到的实际问题，希望对一线工程师们能够有所帮助。

陈臻

小米科技后端工程师，技术深度折腾人员。坚信科学，分享技术。

博客地址：http://www.54chen.com。

淘宝Tengine

易运维的高性能Nginx服务器

文 / 朱照远，姚伟斌

Tengine的由来

Nginx是近几年脱颖而出的一个非常优秀的Web服务器，它以资源消耗低、并发能力强著称，现在是世界上第三大Web服务器。在淘宝，我们用它来服务静态文件、PHP动态页面，做反向代理和负载均衡等。根据淘宝的实际需求，我们开发了数十个不同用途的模块。但随着使用的增多，它的一些不足和有待改进的地方也逐渐凸显。例如，Nginx不支持动态模块加载，不同的应用往往需要编译不同的RPM包，从而导致运维比较麻烦；Nginx欠缺输入请求体过滤器机制，从而使得开发安全模块比较困难；不支持Syslog的方式发送日志，导致日志管理烦琐等。Nginx缺少的这些功能都不能通过开发第三方模块来实现，因此我们开始对它的核心进行深度定制和开发。另外，我们在Web服务器领域也积累了一些经验和创新性的想法，希望在Nginx优秀的基础上，继续加强它的性能、安全和可运维性。这就是Tengine项目的由来。

Tengine是Nginx的一个超集，它基于Nginx的最新稳定版本，对其核心进行扩展和增强，同时保持对Nginx的100%向后兼容性。使用Nginx为Web服务器的业务可以无缝迁移到Tengine。因为Tengine继承了Nginx的优点，所以相对于Apache这样传统的Web服务器，它性能更高，而资源占用（CPU、内存等）更省。在处理大量并发的请求时，它的表现更出色、稳定。同时，Tengine经受住了淘宝生产线的长时间考验，对于访问繁忙和服务器数目众多的大型网站尤为适合。

基于生产环境的实际需求，我们跟淘宝的运维工程师紧密合作，对Tengine进行开发，因此我们设计出来的模块也更着眼于实用性、可用性和运维性。例如我们开发的动态模块加载功能，就可以免去打包和编译的烦琐工作，让Nginx使用第三方模块像使用Apache一样方便。Tengine的命令行，可以显示编译进去的模块和全部支持的指令。Tengine的Syslog功能，可以支持Syslog、Pipe、File等多种记录方式，相当灵活。结合tsar开发的统计模块，甚至可以统计QPS、响应时间等数据。Tengine的主动式健康检查模块，可以在不改动配置的情况下，感知后端服务器的健康状况，主动屏蔽有问题的服务器。

2011年12月初，Tengine正式开源。项目主页在http://tengine.taobao.org。淘宝之所以开源Tengine，是因为淘宝是开源软件的受惠者，公司一直很支持技术项目的开源以回馈开源社区——通过Tengine的开源，我们希望能帮助和淘宝一样对高性能Web服务器有迫切需要的人或互联网公司，大家一起享受开源带来的技术进步。同时，开源对于Tengine本身的发展也更有利，例如我们可以获得更多用户的意见和建议、Bug反馈甚至是Patch等。现在Tengine在国内和国外有很多用户，各项功能有众多线上系统在使用，因此他们的反馈也让Tengine更加稳定和注重实效。自开源以来，我们每隔一个月左右发布一个新版本，添加和修改一些功能及修复Bug。当Nginx本身升级时，Tengine也会定时合并Nginx的更新。同时我

们也在和Nginx公司合作，将Tengine对于Nginx的改进提交给他们。最近我们翻译了部分Nginx的英文文档，被Nginx官方收录。他们对Tengine的一些功能也表示了浓厚的兴趣，因此Tengine中的部分功能有望在不久的将来出现在标准Nginx中。

Tengine的改进

Tengine目前的一些性能改进如表1所示。

表1　Tengine对Nginx主要改进模块

应用模块	concat，user_agent，footer，slice
upstream模块	upstream_check
框架模块和Web开发	Lua
管理模块	backtrace，sysguard，traffic status
核心补丁或模块	dso，input body filter，syslog，CPU affinity，procs
数据结构	4-heap，trie

计时器优化

Timers（计时器）是网络服务器中一个很重要的基础设置，用来管理读写超时和应用逻辑的超时等。其常见操作有添加超时、删除超时以及查找最小的超时值。Nginx使用Red-black tree（红黑树）作为其计时器的数据结构。红黑树对应于添加、删除和查找最小值的算法复杂度都是O(log n)。在Tengine中，我们将Nginx的计时器数据结构改为了4-heap（四叉最小堆）。四叉堆是二叉堆的变种，比二叉堆有更浅的深度和更好的CPU Cache命中率。最小堆的添加、删除的复杂度和红黑树一样都是O(log n)，但在查找最小值时，它的算法复杂度是O(1)，即只要取出堆顶的第一个元素即可，因此比Nginx的红黑树更适合频繁获取最小值的场景，特别是在处理大量连接时，用最小堆性能提升比较明显。

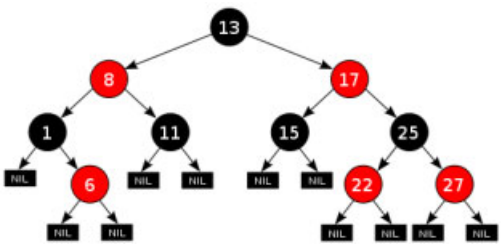


图1　红黑树（图片来自wikipedia）

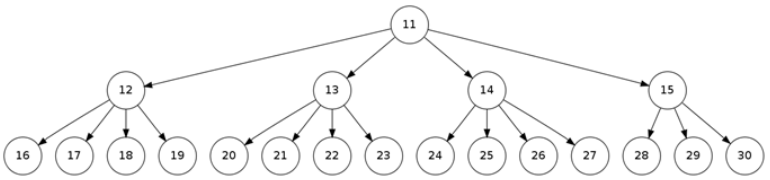


图2　Tengine使用四叉最小堆代替红黑树

浏览器和爬虫的判断优化

判断浏览器的类型是Web服务器的一个常见需求。Nginx中判断浏览器的方法是对关注的浏览器种类在User-Agent头中做暴力查找（strstr）。strstr本身的算法复杂度是O(n^2)，Nginx查找的是多个串，因此其最终算法复杂度是O(n^3)。随着现在移动端的浏览器增多，原有模块的复杂度成指数增长，性能不高。在Tengine中，我们开发了一个全新的user_agent模块，使用了trie（前缀树）来搜索多个可能的浏览器匹配串。它将所有的匹配字符串构造出一个自动机，每次匹配，它的算法复杂度只需要O(n)。因此复杂度不会随着匹配串数量的增加而增加。

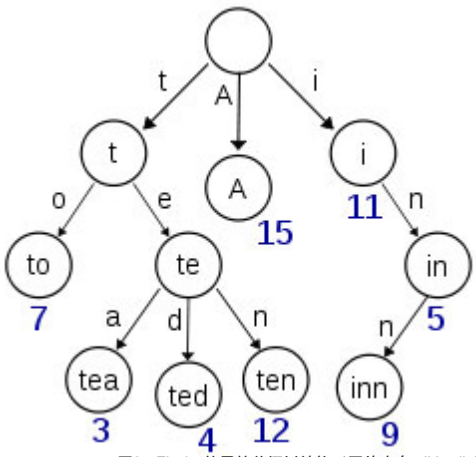


图3　Tingine使用的前缀树结构（图片来自wikipedia）

自动绑定CPU亲缘性

原有的Nginx CPU绑定需要手工操作，在Tengine中我们将Worker进程和CPU进行自动绑定，可以减少因CPU的Cache失效带来的性能损失，从而提高性能。另外，这样也减少了运维配置的工作量。

Tengine对Nginx机制的增强则包含以下几个方面。

Lua模块

基于降低Nginx模块开发难度的初衷，Lua模块（ngx_lua）将Lua嵌入进Nginx核心中，借助

于Lua的协程和Nginx的事件模型实现同步、非阻塞的I/O操作，开发者在Nginx配置文件中可串行同步编写Lua脚本来处理业务逻辑，既可以用它来黏合各种上游（Proxy、Drizzle、Redis、Memcached等）的输出，也可以使用它的Cosocket接口来编写访问上游的客户端。得益于Lua解释器极低的开销和JIT技术（LuaJIT），用户不用编写复杂的C模块就能获得极高的吞吐性能。也可以动态更改逻辑，不用再重新编译Nginx代码，从而带来了极大的灵活性。

Lua模块在初始化时为每个Nginx工作进程创建一个Lua/LuaJIT实例（Lua VM），同一进程处理的所有请求将共享该实例，并且Lua模块将用户Lua代码包装为协程工厂缓存在Nginx内，一个请求到来时协程工厂为它分配一个独立协程来运行业务逻辑。在需要进行阻塞的I/O操作时，Lua模块自动将I/O操作委托给Nginx的事件处理模型，并保存正在运行的协程上下文，返回到Nginx工作进程中处理其他请求，等到I/O操作完成时，又会恢复该协程继续运行。

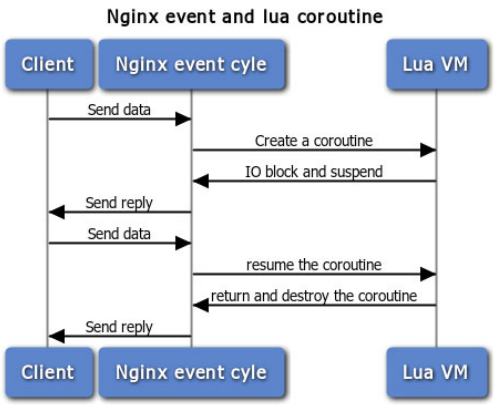


图4 Nginx事件和Lua协程

动态模块支持

Tengine中加入了动态模块功能，对模块实现了动态编译，加入模块不再需要静态编译整个Tengine代码。使用方法类似Apache，在使用时可以当场动态编译想加入的模块，非常方便。

1. 我们提供类似apxs的编译工具，将模块编译成动态链接库。

2. 在Tengine启动时通过动态链接库读入模块的模块结构体，这个结构体包含了模块处理的所有信息。

3. Tengine有内置的模块加载顺序表，也可在配置文件中显式的指定模块的加载顺序，保证模块加载顺序正常。

4. Tengine内部通过两个版本号（Major和Minor）来控制动态链接库（.so文件）的前后兼容性。当Major版本号相同时，较新版本的Tengine兼容较旧版本的.so文件（Tengine的Minor大于.so文件）。只有当Tengine的API发生重大变化时，Major的版本号才发生变化。增加新的API只会增加Minor版本号。

输入体过滤器支持

Nginx没有对请求主体内容的过滤机制，而且在处理较大请求时，可能会缓存到磁盘的临时文件上，因此对输入体的分析和过滤很不方便。Tengine中增加了对于读取用户请求输入体的回调函数，该函数优先于缓存磁盘执行。在收到请求体时会调用这个回调函数，可以方便地对上传的内容进行过滤。而且所有输入体过滤器以链式流程处理。

开启额外进程的机制

Tengine中可以方便地启动进程，这些进程可以独立于原有Nginx工作进程，用来执行某些特殊逻辑（例如非HTTP的应用场景）。该机制在Tengine中增加了一种全新的模块类型，可以开发多个不同用途的进程模块。

对Syslog和管道日志的支持

Syslog功能对于集中式的日志管理非常有用，因此现有大部分的服务器软件都支持Syslog功能。Tengine可以将错误日志和访问日志发送到本地或远程的Syslog服务器。我们完全实现了底层Syslog的协议（使用UDP），解决了Syslog接口阻塞的问题。Tengine也支持通过管道方式将日志写到另一个程序，如Cronolog。此外，在Tengine中，还可以对日志进行抽样，例如只记录1%的日志，从而降低对磁盘I/O的压力，对繁忙的业务颇有用处。

API的增强

Tengine对Nginx的API进行了扩充，如内存操作、HTTP头处理等，以简化模块开发的难度。

目前Tengine比Nginx增加功能模块主要有下面一些。

■ Concat模块

可以组合多个JavaScript和CSS请求变成一个，从而降低下载时间，提高用户体验。该模块对于提高前端的响应时间非常有用。

■ Sysguard模块

在系统的Load或者内存（Swap）使用超过一定阈值或比例时返回等待页面，从而保护服务器。

■ User_agent模块

利用trie结构，扫描浏览器和爬虫的种类，定义$browser和$os变量，比Nginx的Browser模块更加灵活且性能更高。

■ Footer模块

在响应内容后添加一段内容。可用来添加Host信息，对定位大量服务器中出问题的个别机器很有用。

■ Slice模块

用来访问一个文件中的一个片段，可以指定开始和结束的偏移值也可以增加头和尾。

■ Backtrace模块

在遇到异常如崩溃时将调用栈输出到日志以便于问题定位。

■ 主动式Upstream健康检查模块

可以对后端的HTTP、HTTPS、MySQL等类型的服务器定期发起心跳包，维护后端的健康情况，当服务器不可用时，就不再向其发送实际请求。实现了Tengine与后端服务器的高可用。

针对易运维性，我们在几个方面对Tengine进行增强，表2是Nginx与Tengine的对比。

Tengine目前正在做的改进还包括以下方面。

■ 负载均衡的增强如更多负载均衡算法和云的支持等，一致性Hash模块、Session保持模块、后端连接数限制模块、随机负载均衡模块等。

■ Cache功能增强，主要是内存Cache的支持，降低高并发对磁盘I/O带来的影响。

■ 更强的统计模块，可以根据端口或者域名统计流量、连接数等有用的信息。

表2　针对易运维性，Tengine所做的改进

功能	Nginx	Tengine
日志	File	Syslog, Pipe, File，可抽样
编译方式	每次静态编译	核心模块静态编译，功能模块各自编译，动态加载
健康检查方式	被动式检查	主动式心跳检查
状态统计	只显示并发连接数和请求书	可针对域名、端口进行统计，也可对流量进行统计
后端连接数限制	无	可以针对后端服务器进行
连接数限制	命令行可显示版本号和编译	选项还可以显示所以编译进去的模块（-m）、所有支持的指令（-l）、输出所有include 的文件（-d）
CPU 亲缘性	手工绑定	自动绑定
过载保护	无	Sysguard 模块
崩溃输出堆栈	Coredump	Coredump、backtrace 模块

Tengine的社区化发展

目前Tengine主要由淘宝核心系统部维护，其他部门如量子团队、系统保障部等工程师的参与也非常活跃。前淘宝工程师章亦春（agentzh）也给Tengine贡献了大量代码。此外，国内其他互联网公司如搜狗等，也开始参与Tengine的合作开发。

当然，Tengine有着出色表现的最主要原因是我们站在了Nginx这个巨人的肩膀上——正是因为Nginx创立者Igor Sysoev良好的架构设计，优雅的编程风格，对细节的完善处理，让我们受益匪浅，在此我们要对Igor Sysoev致以最高的敬意。

朱照远

花名叔度。淘宝网高级技术专家，任职于核心系统部服务器平台组，Tengine团队成员，负责淘宝Web服务器的开发与Web平台设施的搭建。爱好为开发高性能服务器和Linux内核网络协议栈优化。

姚伟斌

花名文景。淘宝系统工程师，任职于核心系统部服务器平台组，负责淘宝Tengine项目的开发，致力于让Web服务器更好、更快地服务广大用户。

利用Puppet导出资源特性自动配置系统

导出资源是Puppet中非常强大的特性，它可以让我们将定义在一台主机上的资源拿到其他主机上使用。例如，它可以让一个由Puppet管理的负载均衡器获知每一个可用的后端。Puppet在配置运行时收集并存储这些资源，然后在其他主机需要时将这些资源及相关信息提供给它们。

当Puppet master开启配置存储后，我们就能将资源从一个节点的编译目录中导出了。这些导出资源随后可以在另一个节点中被收集，这样节点间就可以动态地、自动化地交换配置信息了。

在本文中，我们将看到三个使用虚拟资源的例子：第一个例子将从每一个Puppet管理的节点中导出SSH主机公钥，并将这些资源集中存储在配置存储数据库中。这样每一个节点就能收集其他所有节点的主机公钥了。这一配置可以提高安全性并消除第一次使用SSH登录时常见的“unknown host”警告。第二个例子使用导出资源在上线新的Puppet Master后端后动态地重新配置一个负载均衡器。最后，你将看到如何动态和自动化地重新配置Nagios监控系统。

使用导出资源

当Puppet Master开启配置存储后，我们就能将资源从一个节点的编译目录中导出了。这些导出资源随后可以在另一个节点中被收集，这样节点间就可以动态地、自动化地交换配置信息了。本节我们将展示一些常见的导出资源例子。

第一个例子将从每一个Puppet管理的节点中导出SSH主机公钥，并将这些资源集中存储在配置存储数据库中。这样每一个节点就能收集其他所有节点的主机公钥了。这一配置可以提高安全性并消除第一次使用SSH登陆时常见的“unknown host”警告。

我们提供的第二个例子使用导出资源来在上线新的Puppet Master后端后动态地重新配置一个负载均衡器。

最后，你将看到如何动态和自动化地重新配置Nagios监控系统，以检查新上线的由Puppet管理的系统的可用性。

自动化的SSH主机公钥管理

当一个大的网络中有新的系统上线后，其他系统的known_hosts文件就过时了，这会导致SSH登录时的“unknown host”警告。对于这个问题，Puppet使用配置存储和导出资源提供了一个简单而优雅的解决方案。在新的系统上线后，Puppet将更新所有其他系统上的known_hosts文件，追加新的系统的主机公钥。这种对known_hosts文件的自动管理同时降低了被注意不到的“中间人”攻击的可能性，从而提高了安全性。

我们知道，任何一个资源都可以通过在声明前使用@符号被声明为虚拟的。而当资源需要被声明为虚拟的并使用配置存储导出到其他节点时，我们使用另一个类似的语法，@@。使用@@可以让任何一个节点的配置目录收集这个资源。代码1展示了如何通过这个语法导出SSH公钥。

```
class ssh::hostkeys {
  @@sshkey { "${fqdn}_dsa":
    host_aliases => [ "$fqdn", "$hostname", "$ipaddress" ],
    type         => dsa,
    key          => $sshdsakey,
  }
  @@sshkey { "${fqdn}_rsa":
    host_aliases => [ "$fqdn", "$hostname", "$ipaddress" ],
    type         => rsa,
    key          => $sshrsakey,
  }
}
```

代码1　导出ssh密钥资源

这段Puppet代码和我们目前使用的有些不同。

为了让网络中所有节点的SSH主机公钥可以被导出和收集，需要将类ssh::hostkeys包含进它们的配置目录中。所有的资源和参数都由Facter的fact值提供的变量进行设置。在代码1中，带有@@符号的两个sshkey资源被声明为虚拟资源，并被导出到中心配置存储数据库。每个资源的标题包含了后缀_dsa或_rsa，以防止它们之间互相冲突。为了确保整个网络中的每一个资源都有一个唯一的标题，标题同时包含了导出主机密钥的节点的正式域名。

参数host_aliases提供了其他可以用来访问这个节点的名字和地址。当从其他系统连接到这个节点时，这一信息对于防止“unknown host”的警告非常重要。在这个例子中，我们提供了正式域名、短的主机名以及系统的IP地址。这些值都来自于自动创建的Facter变量。

参数type和key提供了关于公钥的信息。$sshdsakey和$sshrsakey的值来自于Facter并且在每一个主机上都是可用的。

导出这两个sshkey资源并不足以在每一个节点上配置known_hosts文件。我们还必须像代码2一样为Puppet收集所有被导出的sshkey资源，以便对known_hosts文件进行完全管理和及时更新。

```
class ssh::knownhosts {
  Sshkey <<| |>> { ensure => present }
}
```

代码2　收集导出的sshkey资源

类ssh::knownhosts需要被包含在所有由Puppet管理的SSH的known_hosts文件的节点配置目录里。要注意的是我们使用了双尖括号来从配置存储数据库收集资源。这与收集虚拟资源非常相似，不同的是虚拟资源使用的是一个单尖括号。我们同时指明了在收集导出的sshkey资源时ensure参数采用值“present”。

注意，在一个资源被收集时指定额外参数的能力是在Puppet 2.6.x及以后的版本中新增的，对于Puppet 0.25.x及更早的版本来说不存在这项功能。

当这两个类配置好并被添加到网络中所有主机的节点分类中后，操作员就可以验证网络中每一个节点的主机密钥是否已被收集了。

首先，我们的操作员运行了位于mail.example.com主机上的Puppet agent。因为这是第一台运行Puppet agent的主机，他只期望能够收集两个SSH密钥——这两个密钥均由mail主机自己产生，如代码3所示。

```
# puppet agent --test
info: Caching catalog for mail.example.com
info: Applying configuration version
'1293584061'
notice: /Stage[main]//Node[default]/
Sshkey[mail.example.com_dsa]/ensure: created
notice: /Stage[main]//Node[default]/
Sshkey[mail.example.com _rsa]/ensure:
created
notice: Finished catalog run in 0.02 seconds
```

代码3　mail.example.com上运行的第一个Puppet agent

注意这两个从配置存储数据库中收集的sshkey资源，它们是从mail.example.com主机导出的SSH DSA和RSA公钥。

在代码4中，操作员在Web服务器上运行了Puppet，期望web和mail这两台主机的公钥都能够被收集。

```
# puppet agent --test
info: Caching catalog for web.example.com
info: Applying configuration version
'1293584061'
notice: /Stage[main]//Node[default]/
Sshkey[mail.example.com_rsa]/ensure: created
notice: /Stage[main]//Node[default]/
Sshkey[mail.example.com_dsa]/ensure: created
notice: /Stage[main]//Node[default]/
Sshkey[web.example.com_rsa]/ensure: created
notice: /Stage[main]//Node[default]/
Sshkey[web.example.com_dsa]/ensure: created
notice: Finished catalog run in 0.43 seconds
```

代码4　web.example.com上运行的第二个Puppet agent

如代码4所示，位于web.example.com上的Puppet agent管理了总共四个ssh主机密钥资源。来自mail主机和web主机的rsa和dsa密钥都被导出并存储在配置数据库中。

最后，再次在mail.example.com上运行Puppet agent将使从web主机导出的两个公钥被收集和管理。代码5展示了操作员如何验证这一点。

```
# puppet agent --test
info: Caching catalog for mail.example.com
info: Applying configuration version
'1293584061'
notice: /Stage[main]//Node[default]/
Sshkey[web.example.com_rsa]/ensure: created
notice: /Stage[main]//Node[default]/
Sshkey[web.example.com_dsa]/ensure: created
info: FileBucket adding /etc/ssh/ssh_known_
hosts as {md5}815e87b6880446e4eb20a8d
0e7298658
notice: Hello World!
notice: /Stage[main]//Node[default]/
```

```
Notify[hello]/message: defined 'message' as
'Hello World!'
```

代码5 mail.example.com上运行的第三个Puppet agent

otice: Finished catalog run in 0.04 seconds如同期望的一样，这两个从web主机导出的SSH公钥资源被收集到了mail主机。通过导出和收集这两个sshkey资源，即使有新的主机被添加到这个网站中，Example.com的员工也可以使所有主机自动获知其他主机的身份。只要Puppet频繁运行，每一个系统的known_files文件都会包含网络中其他每一个系统的公钥。

在下一个例子中，你将看到如何利用这个特性自动添加后端节点到一个负载均衡池。

导出负载均衡器后端资源

在上一个例子中，SSH公钥资源被导出并存储在配置数据库，以便网络中的每一台主机都能收集其他主机的公钥。但在一个更小的规模上，你可以用同样的方法将资源导出到网络中一个单独的节点，例如一个负载均衡器上。

在这个例子中，HTTP后端节点将只导出被负载均衡器收集的配置资源。这种组合方式避免了每次向网络中添加一个新的后端时都要对负载均衡器做手工的重新配置。

每一个负载均衡器的后端将导出一个定义资源类型来代表负载均衡器的配置。现在让我们看一下Example.com的操作员如何配置这个系统。这个例子中使用的负载均衡器是Apache。Example.com的操作员使用一个放置在/etc/httpd/conf.d.members/目录的文件片段来模块化一个HTTP后端的配置。我们先来看一下定义资源类型，见代码6。

```
define balancermember($url) {
  file { "/etc/httpd/conf.d.members/worker_${name}.conf":
    ensure => file,
    owner => 0,
    group => 0,
    mode => "0644",
    content => "  BalancerMember $url \n",
  }
}
```

代码6 负载均衡器后端的定义资源类型

这个配置文件片段只包含了一行内容，一个负载均衡器池成员的URL。使用一个定义资源类型是推荐做法，因为当这个资源类型被导出时，所有在它内部声明的资源都会同时被导出。

负载均衡器的配置和扩展Puppet所使用的Apache配置非常相似。如果不使用导出资源，Example.com的操作员就只能静态地定义他的负载均衡器配置，如代码7所示。

```
<Proxy balancer://puppetmaster>
  BalancerMember http://puppetmaster1.example.com:18140
  BalancerMember http://puppetmaster2.example.com:18140
  BalancerMember http://puppetmaster3.example.com:18140
</Proxy>
```

代码7 负载均衡器的前端配置

在这个例子中，我们静态定义了三个Puppet master后端。如果Example.com的操作员想要添加另外的后端，他只能向这个Apache配置段添加第四行配置。

导出资源可以让他省掉这一手工步骤，并且在一个新的工作节点上线并被Puppet配置后自动添加对应的配置。为了做到这一点，Example.com的操作员使用一条Include语句替换掉了所有的BalancerMember语句，这条Include语句将读进所有的文件片段。在Puppet清单中，这些配置语句使用balancermember定义类型进行模块化，如代码8所示。

```
<Proxy balancer://puppetmaster>
  Include /etc/httpd/conf.d.members/*.conf
</Proxy>
```

代码8 在负载均衡器配置中包含导出的文件片段

只要配置Apache包含所有位于conf.d.members目录的文件，Example.com的操作员就不再需要手动添加每一行配置了。相反，他使用导出资源来配置Puppet管理这些独立的文件片段。

导出每一个负载均衡器成员的Puppet配置和我们在SSH主机密钥那个例子中见到的非常相似。Puppet配置非常简单，每一个工作节点需要为它自己导出一个单独的balancermember资源：

```
class worker {
  @@balancermember { "${fqdn}":
    url => "http://${fqdn}:18140",
  }
}
```

注意，Example.com的操作员使用了正式域名作为资源的标题。这样做能保证不会有重复的资源声明，因为每一个后端都有一个唯一的fqdn fact值。使用这种方式声明的定义资源将导出两个资源到配置存储数据库，balancermember资源以及代码6所示的包含在它内部的文件资源。这两个资源中的任何一个都不会在后端节点上面被收集。

自动配置的最后一步是需要Example.com的操作员在负载均衡器节点收集所有这些被导出的资源，如代码9所示。

```
class loadbalancer_members {
  Balancermember <<| |>> { notify => Service["apache"] }
}
```

代码9　收集被导出的负载均衡器后端成员

使用双尖括号语法的操作符从配置存储数据库收集所有的balancermember资源。另外，他还使用了一个参数块来将Puppet对balancermember资源所做的所有变更通知Apache服务。就像虚拟资源一样，可以指定一个参数块来向被收集的资源添加额外的参数。这个语法是Puppet 2.6.x新增的，之前的版本不能与被收集的资源创建相互关系。

在这个例子中，我们看到了一个使用Apache的Include语句的简单版本的文件片段样式。Web服务器的后端节点使用一个定义资源类型可以轻易地模块化它们的Puppet配置。使用一个定义资源类型，Example.com的操作员导出了负载均衡器的相关资源，用于当新的后端上线时自动重新配置前端的负载均衡器。

下面，你将看到当有新的主机添加到网络时，导出资源如何自动重新配置一个中心的Nagios监控系统。

自动化的Nagios服务检测

到目前为止，你已经看到导出资源是如何在新的机器上线时使用Puppet自动重新配置Example.com的系统了。包括如何自动化管理SSH已知主机密钥来提高安全性，以及当新的后端被添加到一个负载均衡器池时如何自动化地重新配置Apache。

在最后一个导出资源的例子里，你将看到Example.com的操作员如何配置Puppet来自动监控新上线的系统。监控服务可用性是所有网站都要面临的问题。Puppet能帮助我们快速和简单地解决这个问题，同时减少管理监控系统自身所需要的时间和精力。

这个例子特别集中在Nagios上。Puppet内置了Nagios的原生类型和提供者，不过，这一节的概念同样适用于任何一个在新的主机上线并需要被监控时，需要对中心系统进行重新配置的监控软件。

在Nagios中，执行服务检查的系统叫做监控系统。运行在监控系统上的Nagios服务查看/etc/nagios目录中的文件来挑选出需要被监控的目标系统。Example.com的操作员希望Puppet能够在有新的目标系统上线时自动重新配置监控系统。

为了达成这一目标，Example.com的操作员首先在Puppet中定义了两个类。第一个类叫做nagios::monitor，用于管理Nagios服务并收集由nagios::target类导出的服务检查资源。现在让我们来看一下这两个类（见代码10）。

```
# Manage the Nagios monitoring service
class nagios::monitor {

  # Manage the packages
  package { [ "nagios", "nagios-plugins" ]: ensure => installed }

  # Manage the Nagios monitoring service
  service { "nagios":
    ensure     => running,
    hasstatus => true,
    enable     => true,
    subscribe => [ Package["nagios"], Package["nagios-plugins"] ],
  }

  # collect resources and populate /etc/nagios/nagios_*.cfg
  Nagios_host    <<||>> { notify => Service["nagios"] }
  Nagios_service <<||>> { notify => Service["nagios"] }
}
```

代码10　/etc/puppet/modules/nagios/manifests/monitor.pp

如同你所看到的，Example.com的操作员配置了Puppet来管理Nagios包和服务。类nagios::monitor需要被包含在监控节点的配置目录中。除了软件包和服务外，还有两种额外的资源类型从配置存储数据库被收集过来，它们包括了所有的nagios_host和nagios_service资源。

当收集这些主机和服务资源时，操作员还添加了notify元参数来确保当新的节点导出它们的信息到配置存储数据库时，Nagios监控服务能自动重载配置。

注意：更多关于nagios_host和nagios_service这两个Puppet资源类型的信息可以在网上找到。除了这两个基本的服务检查类型，还有很多其他Nagios管理相关的资源类型。如果你需要让Nagios意识到主机间的相关性，以减少服务中断时产生的报警，或者需要管理自定义的Nagios服务检查和命令，请查看最新的Puppet类型参考：http://docs.puppetlabs.com/ references/stable/type.html。

Example.com的操作员使用nagios::target类来配置Puppet导出Nagios服务和主机资源。如代码12，这个类只包含了导出资源。除非像代码10一样对它们进行收集，否则这些资源不会在任何节点上被管理。

```
# This class exports nagios host and service check resources
class nagios::export::target {

  @@nagios_host { "$fqdn":
    ensure  => present,
    alias   => $hostname,
    address => $ipaddress,
    use     => "generic-host",
  }

  @@nagios_service { "check_ping_${hostname}":
    check_command       => "check_ping!100.0,20%!500.0,60%",
    use                 => "generic-service",
    host_name           => "$fqdn",
    notification_period => "24x7",
    service_description => "${hostname}_check_ping"
  }

}
```

代码12 /etc/puppet/modules/manifests/target.pp

在代码12中，Example.com的操作员配置了两个导出资源，其中一个向监控节点提供了目标主机自身的信息。这个资源在收集这些资源的节点上定义了一个Naigos主机，位于/etc/nagios/*.cfg。nagios_host资源的标题被设为fact值$fpdn。使用正式域名作为资源标题是为了确保配置存储数据库中没有重复的资源。另外，操作员还为目标主机增加了一个别名，使用了由fact值$hostname提供的短主机名。最后，目标节点的地址设置为来自Facter的$ipaddress变量。

在一个描述目标主机的资源被导出时，操作员同时为这个主机导出了一个基本的服务检查。如同我们所看到的，这个服务检查会对目标节点执行一个基本的ICMP ping命令。这个资源的host_name参数同样来自于Facter的fact值$fqdn。check_command参数看起来让人有些困惑，不过这样是正确的，它直接使用了Nagios的配置文件语法。从左到右阅读check_ping这一行，它可以解释为在ping命令花费超过100ms或者有20%的丢包时，Nagios将发出一个警报。而当ping命令花费超过500ms或者有超过60%的丢包时，Nagios则会发出一个严重警报。警报发出的时间段为每天24小时，每周7天，这是Nagios默认配置的警报时间段。最后，操作员还使用由Facter设置的短主机名为服务设置了一个描述性的标签。

当Example.com的操作员将新的系统上线后，他唯一要做的只是确认这些新系统的配置目录中包含了nagios::target类，然后Puppet就会自动做好对中心Nagios监控系统的重新配置。另外，如果操作员需要更多的系统来监控所有这些节点，他只需要将nagios::monitor类包含在新增加的监控节点的配置目录中，然后这些节点就会自动从配置存储数据库中收集所有的主机和服务资源。

本文节选自《精通Puppet配置管理工具》，感谢人民邮电出版社图灵公司授权。本文在节选时有删节。

一分钟先生

如何从技术岗位走向管理岗位

技术人员如何华丽转身走向管理岗位是很多人关注的话题。本期三位嘉宾将分别从如何做准备、如何转换思维、如何胜任管理岗位这三个方面来解析这一话题。

CTO 俱乐部 会员

黄峥嵘

维苏威高级陶瓷（苏州）有限公司亚太区IT应用经理

机会总是留给有准备的人

机会总是留给有准备的人。在被从技术岗位提拔到管理岗位之前，技术人员就要具备管理岗位所需要的基本素质和能力，将功课做在前面，提拔只是最后一步。然而，从技术岗位走向管理岗位需要具备哪些素质呢？我结合自己十年的工作经验谈谈自己的看法。

■ **做好本职工作是第一步**。技术人员能管理好自己，出色完成本职工作是第一步。在领导交给你任务时，要主动制定工作计划，定期向领导汇报工作进展，出了问题及时沟通，且要勇于承担责任，同时确保工作顺利进行。如果能让领导对你的工作完全信任和放心，那么你在自我管理上就已经准备好了。

■ **打好群众基础**。在管好自己的前提下，还要积极帮助周围的同事。在其他同事陷入困境时，即使那件事情与你无关，也要尝试主动伸手援助，这样能帮你赢得同事的信任和尊敬，慢慢在同事中树立的威信。

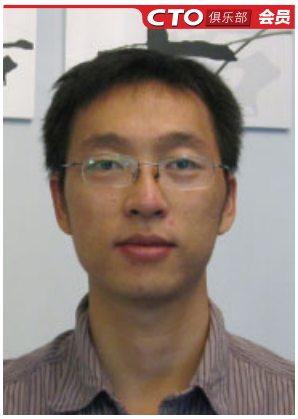
■ **提升思考问题的高度**。这是大多数技术人员最难跨越的一步。技术人员要多学习、多思考，逐渐提升自己思考问题的高度和认识事物的广度。

■ **学习管理知识**。未来你将领导一个团队，你的成功将建立在团队成员成功的基础上，因此要考虑如何领导整个团队取得成功。而“管理”与解决技术问题完全不同，技术知识的对与错能很明显地判断出来，而人与人之间千差万别，如何安排每个人的工作，如何调动他们的积极性，如何处理人员间的冲突，却比较复杂，且不像技术问题那么显而易见，需要细心体会、提前学习。

■ **培养管理能力**。技术人员有时也需要和他人合作完成一个比较复杂的项目，这就是培养管理能力的机会，你要能够很好地与团队成员沟通，主动工作，充分融入团体，与团队成员友好相处，互相帮助。如果每名成员都很尊重你，愿意与你合作，那么未来你成为管理者也就顺理成章了。

■ **沉住气**。如果你既能做好本职工作，又能帮助同事，还能理解领导的思路帮领导分忧，也能带领一个小团队完成团队项目，那么接下来要做的就是耐心等待机会的到来。有时不是领导不想提拔你，而是没有机会，因此千万要耐得住性子、沉得住气。

成功的人永远只是少数。只有1%的人，会在阅读完一篇文章后付诸实践，希望你能成为这1%里的一员，早日完成华丽转身，成为优秀的管理人员。



梁景波
深信服科技研发经理

思维转变是关键

IT公司研发部门的管理人员大多是从公司内部的技术人员中提拔的。在快速发展的公司里，这样的机会更多。然而这种“半路出家”的转型也给我们带来了很多挑战，其中最关键的部分在于思维方式的转变。

■ **从个人成就到团队成就**。无论是做管理还是做技术，成就导向意识是优秀员工的基本素质。只有具备很强的成就导向意识，才能把事情做得超预期，才能追求卓越。

刚刚上任的管理人员的思维方式往往还处于个人成就导向阶段，他们希望向外界发出一个明确的信号，团队之所以取得这样的成绩或者解决某个难题，是因为我的组织和领导。然而这种信号被团队成员多次接到后，会产生功劳都被领导拿走的感觉，从而导致团队的向心力下降。

这时，比较好的做法是：要保护团队成员的成就导向，并且进行鼓励，从而刺激大家的积极性。例如，与某个团队成员共同解决了某个难题后，要弱化自己，重点表扬这个团队成员对问题的贡献。这样再遇到某个难题时，这个团队成员才能保持一样或更高的激情。只有将自己的成就感定位在团队成就上，才能站得更高，避免和团队成员产生直接竞争，从而有效地领导自己的团队。

■ **上下同欲的氛围**。兵法有云：“上下同欲者胜”。一个团队能够健康运作的基础就是“上下同欲”的氛围。要想拥有这种氛围，必须处理好两件事情：发言权和信息透明。

发言权：每个人都希望发表自己对某件事情的看法，尤其是比较关键的事情，并获得聆听。如果管理人员屏蔽这些，所表达的就是一种不尊重。当然不是所有的意见都要被采纳，需要需要合理的决策，但要让大家有发言的机会。

信息透明：这里的信息包括一切可以公开的信息，如上级期望和项目进展等。保持这些信息的透明度，能够提高团队成员对团队绩效的关注度和荣辱感。当大家都站在团队的高度上思考问题时，能够省去很多协调工作。

转型到管理岗位后，就要多花些时间来考虑团队建设问题。只有团队的氛围比较好，才有机会取得较好的成绩。

■ **合理计划，要事第一**。刚刚转型的管理人员往往会有类似这样的抱怨“杂事太多，被不停地打断”。造成这一问题的很大部分原因在于，计划不够周全或者缺乏例行沟通机制。例如，每日站立例会基本上可以消除很多这样的杂事。同时因为管理人员需要接触的人比较多，所以日常处理的事情也会比较多。这与做技术人员有很大不同。

这时就要有非常合理的计划，保证要事被及时处理。需要注意的是，时间的急迫性往往会夸大事情的重要性。如果管理人员总是习惯将任务拖到最后的截止时间处理，则会打乱事情的优先级，导致做事失去计划性。

■ **培养人才，用人所长**。杰克•韦尔奇说过：“在你成为领导之前，成功只同自己的成长有关。当你成为领导之后，成功同别人的成长有关。” 管理人员要把培养下属作为一件重要的事情，只有团队里的人才层出不穷，团队才有能力去不断挑战更高的目标。用人所长则指在工作的安排过程中需要多关注下属的长处，不能过度放大他们的缺点。正所谓“用人所长，天下无不用之人；用人所短，天下无可用之人”。人事任命需谨慎考虑，用对一个人，能省很多心；用错一个人，要操很多心。将合适的人放到合适的位置，是管理人员必须面对的一个难题。

周滨
可口可乐装瓶投资集团中国区CTO

如何成为一名合格的管理者

职业通路是狭窄的，“金字塔”结构很好地描绘了每个人在职场将要走过的路。在职位与薪酬待遇紧密挂钩的当今职场，芸芸技术专家总有一天会面对这个华丽而痛苦的蜕变。能将技术和管理两者有效融会贯通的人才是真正的人才，才是优秀企业激烈争夺的职场精英。本文总结了我多年工作的经验和体会。

大局观

当一名优秀的狙击手成为将军时，首先要克服的问题就是自己射击技术太好，总想亲自上阵。卓越的技术能力成就了今时今日的我们，但在管理岗位上，已经不能再一味地沉迷于对技术的不断追逐。要更清楚地认识到自己的任务已经不再是单纯的技术工作，团队管理的职能已将我们提升到更重要的位置，因此从团队整体出发考虑整个集体的成败得失是管理者应具备的基本素质之一。

当然，作为IT从业人员，技术工作将伴随我们的整个职业生涯，我们仍然可以是一名技术专家，拥有扎实的技术背景，这有助于更好地与队员沟通，更好地理解队员遇到的困难和取得的成就。而卓越的管理能力可以使团队团结一致，为共同的目标不懈努力，取得最后的胜利。因此，在拥有大局观的同时，技术能力和管理能力两者相辅相成、缺一不可。

融合的艺术

IT组织往往是复杂的，通常由各领域的技术专家、开发者、第三方顾问、供应商的实施人员等组成。作为一名管理者，一定要懂得融合的艺术——既能管控和协调各方资源，避免资源浪费，又能有效地组织大家共同前进，提供一个能让每个人充分发挥各自作用的环境。

沟通与演讲

沟通和演讲可能是大多数技术出身的管理者的最痛处。然而，没有良好的沟通和演讲能力，很难成为一个好的管理者。我们不再是在小办公隔断里带着耳机编码一天的工程师，也不再是机房里负责维护设备的系统管理人员。管理岗位需要与人交流，了解工作进展、鼓励队员、讲解公司管理要求等，无一不需要良好的沟通能力。与此同时，向上级汇报工作，展示工作成果，与客户或供应商互动，又毫无例外地需要卓越的演讲能力。

事实上，提高沟通与演讲能力，无非要关注两方面：提高理解别人的能力和增加别人理解认同自己的可能性。这就要不断丰富自己的知识面，与各个层面的人进行交流，多在各种会议中进行演讲，一定能顺利通过这道门槛。

诚信

回想我们当年多么痛恨管理层的“谎言”和从未兑现的“承诺”，因此在走上管理岗位后一定要铭记这点。不要承诺做不到的事情，承诺了就要兑现，给别人期望然后再熄灭它，比没有期望的打击来得更大。没有诚信的管理者所管理的团队是没有战斗力的，自己都做不到的事情，如何期待队员们做到呢?

总结

管理是多个学科和多种能力的结合体。因此，从技术岗位走向管理岗位是一个循序渐进的过程，并非一蹴而就。只有在工作中不断学习和积累，不断完善自己，才能成为一名真正合格的管理者。

破解敏捷团队协作的迷局

文 / 许正华

本文以CA中国技术中心为案例，分析了传统软件组织在实施敏捷方法的过程中所面临的来自团队协作方面的若干实际挑战。并基于对传统式协作与敏捷式协作的根本差别的认识，提出了一套有针对性的改善团队协作的解决之道。

我们的敏捷之路

CA采用Scrum方法实施敏捷软件开发已两年有余，蜜月期早已结束，却并未从变化所带来的初始混乱中走出来。挂在敏捷开发头顶上的光环已渐渐消退，人们丧失了对它的耐心与好感。私下里很多人都在质疑所实施的敏捷软件开发过程相比过去的瀑布式开发真的是一种进步吗?

下面是一些比较有代表性的声音。

- 用户故事应该由产品负责人编写，而不是开发人员或是测试人员。
- 缺乏足够的产品知识。
- 缺乏对设计的评审活动。
- 讨论占用了过多的时间。
- 无法了解设计，因为缺乏相关文档。
- 反馈太少或太迟。
- 开发人员总是延期交付可测试的用户故事。
- 开发人员交付的产品质量缺陷太多。
- 测试人员报告大量低价值缺陷。
- ……

从表面上看，上述声音反映的是软件开发的不同方面，实质上它们都指向了同一个问题——敏捷团队的协作。

“以开发人员为中心”是一种典型的以研发为中心的文化形态，开发人员在协作中扮演了至关重要的角色。而如何在敏捷开发中保持开发人员与产品负责人、开发人员与测试人员、开发人员之间的顺畅协作是一个关乎敏捷实践成败的根本性问题。

敏捷协作与传统协作的区别

并不是只有敏捷软件开发才强调团队协作，因此我们必须更深刻地理解敏捷所提倡的协作与传统的协作之间到底有何区别，以便找出敏捷团队在协作方面所存在的问题，并进行有针对性的改进。

传统式协作

在瀑布式软件开发过程中，协作主要具有以下几个特征。

- 强调清晰的角色定义，即每一个角色会被赋予一系列特定的职责（如表1所示）。

表1　软件团队角色定义（部分）

角色	职责
产品经理	市场调研、竞争分析、需求分析、产品规划和功能设计等
系统架构师	系统业务需求的理解、系统体系架构的创建、技术选型、原型系统的设计与实现等
开发工程师	原型系统的设计与实现、系统设计、软件开发与调试等
测试工程师	测试计划的创建与执行等

- 以流程为驱动。传统的软件开发特别重视流程的制定，每个阶段有明确的输入与输出。
- 以文档为载体的单向流动。各种格式化的文档是信息的载体。虽然也会建立一些反馈机制，但总体上信息是单向流动的。

敏捷式协作

敏捷团队的协作是在信息不断被成员加工并交换的过程中完成的。信息的加工与交换过程不受成员角色、流程与文档的限制，具有不确定性。我把这种新型的信息协作模式叫做“信息激荡”（如图1所示）。理想情况下，一个团队的信息加工与交换活动应该是简短而活跃的。

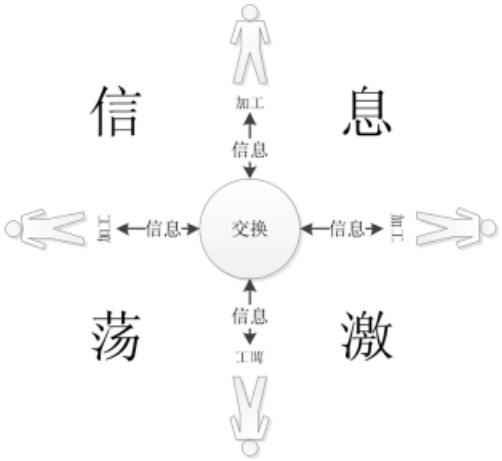


图1 信息的加工与交换

敏捷式协作的产生有其深刻的历史背景：社会生活节奏的加快使得各个行业都要能够更快地对新的信息进行理解、分析并做出决定。

在这个过程中，个人的信息处理能力常常相对不足，需要发挥群体智慧，因为全面的考虑是有效解决问题的关键。传统的基于角色、流程或文档的信息处理方式无法充分激发群体智慧。

这样就不难理解为什么Scrum框架要强调开好四个会议：Sprint计划、每日Stand-up、Sprint评审和Sprint回顾。其目的就是要激发全员的思考潜能，形成群体智慧，纸牌估算是其最典型的代表。

破解之道

四个会议构成了Scrum的有形框架。此外，我们还应该意识到有一些不确定因素在深刻影响着敏捷团队的协作。这其中至少包括“用户故事”、“验收标准”、“任务”和“检查点”等，它们构成了Scrum团队协作的无形框架。

史诗故事的分解

敏捷开发很像田径比赛中的110米跨栏：运动员比拼的不仅是绝对速度，更是良好的节奏。软件开发的节奏问题以往很少被人们提及与重视，这一点在瀑布式软件开发过程中表现得最为充分，敏捷软件开发过程较之则进步了许多。以Scrum为例，产品的每一次发布都包含一系列Sprint，用户故事为团队建立合适的节奏打下了基础。

我们要特别警惕大粒度故事对于团队协作的破坏作用：人们总是希望在一个故事的早期阶段就能够将各种不确定因素消除。团队越不成熟，这种心理预期就越强烈，早期的讨论对项目的影响就越大。不幸的是，大粒度故事会使需求与设计中的不确定因素增多，早期的讨论变得更加困难（理想与现实的矛盾）。理论上，每一个用户需求都可以被一步步细分，直到其尺寸足够小。同时，我们也要警惕其可能产生的副作用：随着需求的分解，每一个故事的用户价值会因为减小而难以把握，完整的用户体验面临着被拆分的风险。

为什么会产生大粒度故事？故事分解绝不是简单地切割蛋糕，它是最高级的软件设计活动之一。它考验的是整个团队对用户需求的理解、产品知识的积累，以及综合设计能力。这些会受以下几方面因素影响。

- 工作地点的分散。要合理分解史诗故事（如图2所示），需要团队成员（产品负责人、开发人员和测试人员等）在前期进行充分的沟通。分散的工作地点将增加沟通成本，降低沟通意愿。
- 技术专家的缺失。我们对相关技术了解得越透

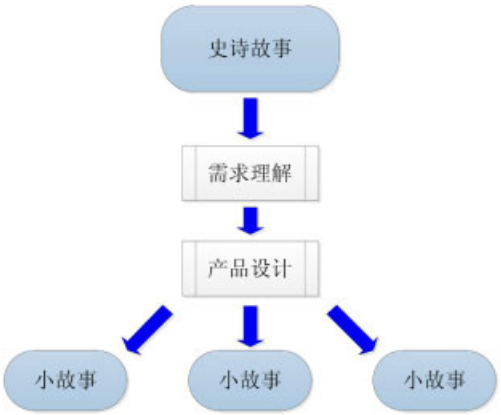


图2 史诗故事的分解

彻，越有可能将大故事分解为小故事。

■ 固有观念的束缚。有些人抱怨用户故事应由产品经理编写，因为他们把用户故事的价值简单理解为对需求的描述，忘记了“**用户故事不是一种设计契约而只是沟通的起点**”；有些人认为故事大一点也无所谓，只要为其多加些任务就可以。

上述因素的共同作用使得两周（开发时间），甚至更大的故事在研发中心是比较普遍的。为了解决这个问题，我们首先从“端正思想”入手，并为团队提供必要的培训，最后为团队建立一般性的指导（例如建议将开发时间控制在一周以内）。

以用户为目标的任务

接下来需要将用户故事进一步分解为更细粒度的工作单元。开发人员依据自己的知识和经验定义不同的开发任务，在这方面他们具有绝对的自由度，外界并不会过多地干预，因为任务通常被视为是内部的、与用户无关的细节。在现实中，“按模块划分”是一种最常被使用的任务划分原则。

与故事分解一样，任务划分也是一种高级软件设计活动，其随意性是开展高效团队协作的障碍之一。敏捷是以用户为中心的，因此我们应该尝试从用户这种统一的视角来划分任务，即“按验收标准划分”。

在图3中，故事A是按照模块划分的任务；故事B是按照验收标准划分的任务。除了以验收标准为任务划分原则，我们还可以创建一些通用的任务（例如，开发一些基础性代码框架）。

与按模块划分任务相比，按验收标准划分任务具有以下明显的优势。

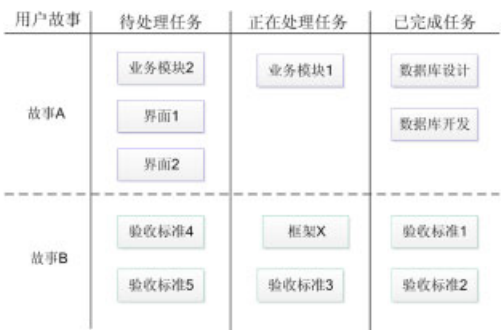


图3 故事A与故事B

获得及时的反馈。每一项验收标准都应是相对独立的，具有一定的客户价值，尺寸更小，且可以测试，因此可以被看成是一个“微故事”。开发团队用验收标准来划分任务可以促进其以更小的粒度将新功能特性交付给测试团队。这样做的好处是显而易见的：我们都知道即使前期对设计进行大量的沟通，也无法取代产品在交付时产生的真实体验。每一次交付可以被看作是一次协作的短脉冲，交付点越多，反馈也就越及时。

提高估算的准确性。造成开发人员估算不准确的主要原因之一是“思考缺乏全面性”。当故事较复杂时，人们为了推进早期设计往往倾向于只关注基本路径，而那些被有意无意所忽略的部分却可能占据大量的项目时间。以验收标准为任务划分原则可以有效地避免遗漏重要的方面，从而提高估算的全面性。

唤醒开发人员的质量意识。开发人员在早期只关注功能实现，到后期对质量进行修补是普遍的现象，特别是在项目时间压力大的情况下。“不积跬步，无以致千里”。产品的质量最终取决于我们每天的工作细节——敏捷开发就是要让我们对待软件开发的态度回归理性。持续地按验收标准进行交付可以确保开发人员对工作结果进行系统级验证，避免“质量炸弹”。

提高例会的沟通效率。“开发人员与测试人员各自描述自己对项目进展的理解，最终让人听得是一头雾水”。这是我们在每日例会上经常遇到的场景。为了提高会议的有效性，需要找到不同角色的共同关注点。验收标准可以帮助团队建立起一个有效的沟通平台。我们甚至可以以此为基础在每日例会上用简短的演示取代枯燥的更新。

请注意：**用户故事的尺寸越大，按验收标准划分任务就显示出越大的优越性。**

带检查点的验收标准

很多人都以“**原则**”和“**实例场景**”来区分验收标准和验收测试。这种划分理论上看似无懈可击，但在应用中却存在着巨大的隐患：验收标准是团队协作的重要工具，它们被认为是对用户故事的良好注解。我们一方面希望将原则与实现细节保

持距离，另一方面也要认识到“单凭对原则的沟通，永远也无法达到对行为的一致理解”。

可能的解决方案之一是扩展验收标准，使其附带一个或多个检查点。传统上，检查点设计只是测试人员的职责与关注点。在Scrum中，我们认为检查点是验收标准的组成部分，需要开发人员与测试人员共同关注并达成一致意见。

带检查点的验收标准实例：

计算机节点导入功能具有可伸缩性（原则）

- 从产品A的管理服务器发现100个计算机节点的时间不超过10秒钟（检查点）
- 选中20个计算机节点存入数据库的时间不超过3秒钟（检查点）

有了检查点，验收测试设计就演变为对检查点的重新组织：一个测试例可以包含一个或多个相关的检查点（如图4所示）。

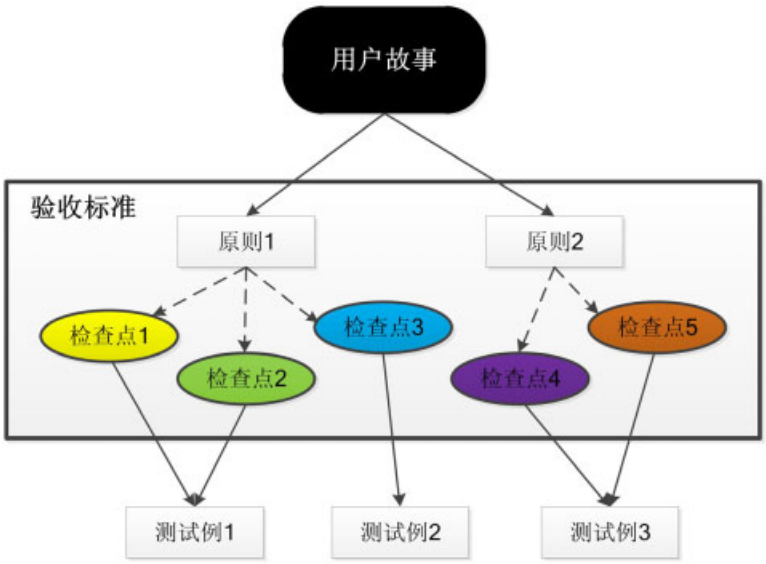


图4　一个测试例可以包含一个或多个相关的检查点

度量团队协作

团队协作水平也是可以度量的。我们设计了一套以验收测试例为基础的度量指标，并追踪其随时间变化的趋势。

测试例总数。每一个基本的测试例都包含了一系列检查点。如果测试例总数可以快速趋于稳定，那么说明前期调研充分、验收标准能够快速地被团队所理解。

已交付测试例。当以验收标准作为任务划分原则时，已交付测试例曲线应持续增加。Scrum团队可以考虑在每日例会上，让开发人员以现场演示替代枯燥的例行更新。

已验证测试例。已验证通过的测试例总数可以持续增加，说明项目风险逐步减小。造成验证速度严重落后于交付速度的一个常见原因是测试人员精力过于分散——每个测试人员要同时服务于若干个独立的用户故事。团队应将同时进行的故事数目控制在一个合理的水平。

首次运行失败（First Run Failure）。战术层面：对首次运行失败率的度量是促进团队合作的一针强心剂。高通过率不仅意味着更少的软件缺陷，而且会使测试团队更加信任开发团队的工作，并激发其参与早期设计讨论的热情。战略层面：降低首次运行失败率是开启调整开发人员与测试人员职责这扇大门的钥匙。开发人员应该更多地承担基本的新功能验证工作。

该方法的最大特点是用对趋势的追踪取代以往对数据简单的统计，它可以持续反映团队成员间的协作状态。

总结

我们渐渐意识到要想让敏捷真正扎根于研发中心，其难度远远超出我们的预期，绝非一日之功，因为这是一场文化的变革。还有很多本文未提及的方面也是值得关注的，例如，团队成员的培养问题：分解故事、编写验收标准、沟通设计、估算时间、控制进度与质量、整理知识这等“琐事”不应只是Scrum Master分内的事情，因为敏捷团队的有效协作最终必须建立在个人良好的自我管理基础之上。人员的问题就留待下次讨论吧！

许正华

目前就职于CA Technologies中国技术中心，从事数据备份和高可用企业软件方面的开发和研究，并专注于软件工程、知识工程、团队文化等领域的研究与实践。

责任编辑：杨爽（yangshuang@csdn.net）

Marty Cagan

过去20多年，Marty Cagan作为负责定义和开发产品的高级经理人为多家一流企业工作过，他亲历了个人电脑、互联网、电子商务的起落沉浮，致力于通过写作、演讲、培训帮助客户打造富有创意的产品。

产品管理模式及组织结构

文 / Marty Cagan　译 / 王瑜洁，潘希颖，姜沈励

Marty Cagan是享有世界声誉的产品管理专家，曾经担任网景副总裁、eBay产品管理及设计高级副总裁。本文是他回顾自己二十多年来从事软件产品管理工作的总结和经验分享，文中分析了最佳产品管理模式及部门组织结构，并对产品副总裁的角色进行了定位。

最佳产品管理模式

看到Google和苹果的成功，我们不免会提出这样的疑问：如何才能像它们那样管理产品？然而，成功是不可以被复制的，盲目模仿可能会酿成大错。

苹果和Google的产品管理模式大不相同，却分别适用于各自的公司（只要它们的创始人仍然致力于打造用户喜爱的产品）。其他的公司尝试这样的产品管理模式却很难成功。想复制苹果的成功，只复制它们的模式不行，还得克隆出另一个乔布斯。

Google的产品管理模式与众不同，但我认为，这种差异化是必须的；苹果也是一样。

当然对产品经理而言有些技巧非常重要，例如评估产品机会、定义产品原则、产品探索，以及原型测试，但企业想获得成功，依靠技巧还远远不够。如同一支球队，球技再好，如果没法保证将球控制在自己脚下，在小组赛中都很难出线。要想赢球，除了球技之外，比赛策略、团队协作、了解对手、适应场地以及良好的状态，一个都不能少。

创建成功的产品管理模式，不仅要提升产品经理的管理技巧，更要确保他们能与团队成员有效合作（这是产品开发团队的管理重点，是产品研发流程的关键），此外还要知道如何创造公司需要的产品类型，清楚如何在市场上击败对手、取得成功。

一般可以通过以下几个方面来选择适合公司的最佳产品管理模式。

- **产品类型**。明确公司的产品类型，是面向大众的网络服务、消费电子设备、企业级软件，还是小型企业服务。每种类型的产品拥有不同的特性，因此要依据不同的产品类型，找出合适的产品管理模式。
- **产品开发过程**。例如，若产品团队使用敏捷方法开发产品，会对产品管理者和设计者有特别的要求。此时，理解产品开发过程非常重要。而每种开发过程都有其局限性，只有打破这种局限性，才能称得上“最佳”。
- **产品管理的角色**。不同公司的责任分工方式不同，而“最佳”产品管理模式能让员工有能力和激情来扮演好自身的角色。这样的模式也同样适用于其他重要的角色，如交互设计师、工程师、投资人和管理层。
- **组织规模**。小型创业团队，资金来源于风险投资，其创始人专注于产品；大型上市公司，拥有大批客户和千余名雇员。两者规模不同，需求也会不同，其管理模式自然不同。
- **企业文化**。有时它是主导因素。例如，在公

司里有一两个人能高效地做出产品决策（并希望将这一状态保持下去），这时产品管理模式要能推动这种状态，而不是阻碍它。

我认为苹果、Google、微软、eBay和Yahoo!的产品管理模式各不相同。那么每种模式还有改进的余地吗？当然！你能从它们那里得到启发吗？当然！但不能强迫公司实行不适合自己的模式，“削足适履”只能成为一个悲剧。

除非公司决定改变企业文化，否则致力于维护企业文化和开发流程中的精华部分会是更好的选择。此外，还要确保所选择的产品管理模式能扬长避短，发挥其积极作用。

重构产品部门组织结构

其实，组织结构并不是我关注的重点，我更关注组织中各个角色及其相关职责。我相信，只要角色定位准确、管理方式得当，大多数的组织结构都是可行的。而且我认为，与其照搬一家成功企业的组织模式，不如根据团队的特点定制一个更加符合企业自身情况的产品组织。

我们都知道，组织结构会影响人们的行为。在其他条件不变的情况下，如果依照下述方式改变公司的组织结构，会收获到意想不到的好效果。

我这里讨论的对象是产品部门，并不涉及公司其他部门，比如销售、客服、财务、业务发展及IT（非产品开发）部门。

我认为CEO、COO，以及子公司的总经理都需要将自己的员工明确地划分至三个部门：营销、产品管理及设计、产品开发。并保证这三个部门同属于公司组织结构的同一层，而不出现其中一个部门隶属于另一个部门的情况。

建议一家中小型公司的CEO/COO，或者大公司的业务经理将组织结构做出如下安排。

- 营销部门包含其企业营销、营销传播、现场营销、产品营销。
- 产品管理及设计部门负责产品管理和用户体验设计（交互设计/信息架构、视觉设计、原型制作、用户研究/可用性开发），如果条件允许，该部门还可引进行业专家。
- 产品开发部门主要负责产品架构、代码编写、质量测试、产品发布、平台运营（SaaS软件运营公司）和项目管理（PMO，项目管理办公室）。

需要补充说明以下内容。

- 为内部员工提供技术服务的IT部门有别于产品开发部门。这两个组织在需求上有本质的区别，因而管理方式也该有所区别。通常是由CIO管理IT部门，而CTO或VP（产品开发副总裁/工程部副总裁）管理产品开发部门。
- 在某些公司将PMO作为最高级别的管理部门，这本身没错。但多数时候，PMO是产品开发部门的一部分，因为项目经理管理的大部分资源都在产品开发部门里，因此产品开发负责人会全权负责产品的实施与交付。但不管在什么情况下，项目经理应直接受项目管理办公室管辖。
- 不管将用户体验设计归入营销部门还是产品开发部门都会出问题。用户体验团队（特别是交互设计师）和产品经理应该是拴在一根绳上的蚂蚱。为了使产品经理与用户体验设计团队的合作更紧密，越来越多的公司选择将产品管理和产品设计合并为同一个部门（但每个职能都有各自的负责人，如产品管理总监负责产品战略，用户体验总监负责平台/软件的整体信息架构及风格）。
- 在营销部门里，通常也会有一些负责营销项目和广告的图形设计师。用户体验组织中的视觉设计团队可能也能满足营销工作的需求，但我觉得在营销部门设立这样的职位会更好。强大的用户体验离不开对交互设计的持续关注和对应用程序的可视化设计。而图形设计师的领导者则应该清楚如何为产品团队提供支持。如果创意总监是产品类视觉设计和营销类视觉设计的双项全能，那么公司就有福了，因为这样的视觉设计团队可以同时满足两种设计需求，所塑造出的品牌形象也会非常一致。但如果视觉设计团队负责人隶属于营销部门却不了解产品的需求，就必须尽快做些改变了。
- 不要把用户研究和可用性研究混为一谈。用户研究是以人口统计学和消费者行为学（包括购买模式）为基础对客户进行分析与研究。因此，只

要产品经理和交互设计师能随时找到用户研究人员，且相互之间建立了良好的工作关系，就可以将用户研究作为营销活动的一部分。而可用性研究是对人物角色、原型及如何能让用户方便地使用产品进行具体的研究。与用户研究不同，可用性的研究成果会直接反馈给交互设计师、视觉设计师和产品经理。因此，可用性研究人员是用户体验设计团队不可或缺的一部分，只有这样，在研究产品的过程中，他们才能更好地与其他成员合作，他们的研究结果才能更好更快地反馈给相关人员。

- 有的公司将产品营销归为产品部门。这种结构不会产生大问题，没准还有一定的优势。但我还是认为将产品营销独立出去会更好一些，因为这样营销人员对自己的角色定位更准确，产品管理和产品营销之间的职能划分也更清晰。即使产品管理和产品营销同属于一个部门，也不应该将各自的职责混为一谈。

- 创业型公司的组织结构通常比较简单，因为公司更看重个人能力。可一旦公司走上正轨，进入快速发展阶段，尽早采用有效的组织结构会是明智之举。

- 如果贵公司的组织结构和我推荐的不同，公司却运转良好，建议保持这种结构不做任何改变。但如果团队挣扎在痛苦的边缘，好产品的诞生比预计的要困难，就可以考虑按照上述方向将组织结构改进一下，看看会不会有所改善。

产品副总裁的角色定位

网站上，公告栏里的招聘信息内列出了大量的产品副总裁或产品总监的职位。这一现象不仅意味着我们的行业正在高速发展，也说明了人们对产品管理的重视程度与日俱增。

我和各种类型的产品公司合作过，为它们推荐并评估了不少产品管理的候选人。在此，我想谈谈对产品管理者的一些看法。

我用“产品副总裁”来指代产品领导者的角色，你可以找到其他类似的称呼，例如产品总监或者首席产品经理。无论哪种称呼，在这里，我指的是公司或者事业部里，最资深的产品管理者。

在公司里，产品副总裁的主要职责是管理产品负责人/产品经理、用户体验设计师，并且直接向CEO汇报工作。多数时候，这个角色与CTO和市场副总裁平级。

做产品副总裁并不容易。称职的产品副总裁会给公司带来显著的改变。优秀的产品领导者倍受追捧，并且常常会自行创业。事实上，只有优秀的产品领导者才可能获得风险投资商的青睐。

具体来说，你所寻找的产品副总裁必须具备以下一些特质。

团队建设能力

产品副总裁的主要职责是组建一支强大的产品经理和产品设计师团队。这意味着招聘、培训、持续性指导是很重要的工作。值得注意的是，管理团队和开发产品完全是两码事，很多优秀的产品负责人和设计师能开发出好产品，却管理不好自己的团队。

提拔一个能力差劲的人到管理岗位无疑是整个团队的梦魇。你肯定觉得没谁会提拔差劲的人，但这种事情时有发生——“这个家伙能力一般，但人际关系不错，并且股东们很喜欢他，我可以让他做领导，然后找个能力强的人在背后支持他。”你真的能指望这样差劲的人可以帮助整个团队提高能力吗？团队的其他成员听到这个消息会怎么想呢？

产品副总裁必须有能力管理整个团队。他必须具备辨别和招募人才的能力，并且能积极持续地帮助下属改正缺点、发挥优势。

产品愿景的规划能力

规划产品愿景就是制定战略来定义和开发产品。愿景能激励全体员工向着同一目标迈进，让公司基业长青。这听起来很简单但做起来棘手。因为面对以下两种不同的情况，需要为团队配备不同的团队领导人。

- 情况一：CEO或者创始人有很清晰的产品愿景。
- 情况二：公司还没有明确愿景，但创始人正在探

索中。

下面是两种比较糟糕的情形。

- 第一种情形是，CEO拥有很强的愿景制定能力，但他觉得需要一个产品副总裁来协助自己（或者董事会希望他雇一个，这种情况更为常见）。CEO希望雇来的产品副总裁与自己有类似的愿景，却往往事与愿违——他们时常会意见不合，导致每位产品副总裁的任期都很短。如果这个职位就像旋转门一样留不住人，结果可想而知。
- 第二种情形是，CEO不擅长愿景规划，但他还是想雇用一个和他有相似愿景的产品副总裁。也许他们不会有分歧（通常他们相处愉快），却会导致愿景的缺失，进而导致产品团队的挫败感，令公司士气低落，缺乏创新。

问题的关键在于产品副总裁需要弥补CEO的不足。如果公司的CEO有明确的愿景，那么很多候选人不太愿意担任这家公司的产品副总裁，他们知道在这样的公司里自己的想法不再重要，只能被动执行CEO的命令。

如果公司有幸拥有一位很有产品远见的CEO，也有一个可靠的产品副总裁来实现愿景，那就再好不过了。可一旦CEO离任，公司将面临极大的风险。产品副总裁难以胜任制订愿景的重任，即使他能胜任，公司的其他部门也未必认可。因此，我通常认为产品创始人最好一直留在公司，即使不以CEO的身份留下。

如果CEO自视很高，认为自己是一个有远见的领导，但其他人并不这么想，怎么办？这就需要一名特殊的产品副总裁，他不但愿景规划能力强，并且心甘情愿而又巧妙地让CEO相信这些愿景都是自己制定出来的。

执行力

不管愿景是谁规划的，如果无法把产品创意从概念变成现实，那么愿景再伟大也是徒劳。产品领导人必须拥有强有力的执行力，口说无凭，要用事实说话。

有很多方法可以让愿景的执行更加持续、快速和高效。产品副总裁不仅要精通当今形势下的产品规划、用户探索、产品探索、产品开发流程，还需要具备很强的执行力，保证团队的每个人都能高效工作。

组织越庞大，产品副总裁的任命要求越严苛，尤其是在董事会管理的体制下。他必须有能力启发并鼓励整个公司向同一个方向前进。

产品文化

好的产品团队需要一帮“牛人”、一个明晰的愿景和强有力的执行力。这些都是树立良好产品文化的基础。

在良好产品文化的熏陶下，团队理解持续快速地测试和学习的重要性；理解成功没有坦途，犯错在所难免，但必须快速改正错误且尽量降低风险；理解持续创新的重要性；知道伟大产品离不开团队合作；尊重设计师和工程师的劳动；知道一个有进取心的团队才能所向披靡。

称职的产品副总裁必须理解产品文化的重要性，最好具备产品文化建设方面的经验，能给出适合公司发展的具体的实施计划。

经验

对于经验的要求，例如行业经验，主要取决于你的公司和行业类型，他最好拥有相关的技术背景，并且对业务和市场了如指掌。

良好的人际关系

此外，产品副总裁需要和其他高层培养出良好的人际关系，特别是CEO和CTO。可以说，良好的人际关系有百利而无一害。面试一结束就开始“培养感情”吧，和CEO、CTO一起吃个饭，聊点儿工作以外的话题。

本文节选自华中科技大学出版社《启示录：打造用户喜爱的产品》一书及作者的博客。该书从人员、流程、产品三个角度介绍了现代软件（互联网）产品管理的实践经验和理念。特此感谢华中科技大学出版社与Marty Cagan先生授权。

专业测试团队会消亡还是新生

文 / 朱少民

敏捷软件开发致使很多人质疑专业测试团队存在的价值，本文对此进行了深度的剖析，并结合技术发展现状给出了软件测试的未来方向。

敏捷软件开发带来的困惑

敏捷软件开发强调“拥抱变化”，认为不能将需求定义一次做到位，也没必要一次做到位，需要不断挖掘，才能逐渐获得真实的需求。这就给测试带来极大的挑战，因为测试需要把验证的标准作为参照系，否则如果需求不清楚，就很难确定测试中发现的问题是不是真正的缺陷，导致测试的设计与执行困难重重。在这种情况下，我们是否只能依赖探索式测试呢?

敏捷软件开发强调持续构建、持续测试。在构建中不仅可以完成单元测试，还可以完成集成测试。因为每天都构建软件包，一旦单元之间（接口）存在问题，就很容易暴露出来。每日构建和集成可被看作持续测试的一种体现。在这种情况下，是否就无需单独的集成测试阶段？测试的投入是否就可以减少呢?

敏捷软件开发强调与用户的沟通和协作，用户起初并不清楚自己的需求，通过软件产品的使用，才逐步明白自己想要什么。如果用户真能参与开发过程，即用户每天都能及时使用正在开发的软件，那么还需要测试人员吗？如果将持续集成和用户及时反馈结合起来，专业测试人员的测试投入是不是就可以大大降低?

敏捷软件开发强调持续交付，即及时发布有价值功能给客户，并尽快地满足客户的需求。之所以能做到持续交付，是因为软件已从产品销售模式转为服务模式——软件即服务（Software as a Service，SaaS），不存在以前产品模式的销售渠道，软件版本的更新只要在自己数据中心的服务器上打个patch就可以了。这种SaaS模式使得持续交付成为可能，软件能够快速上线、用户也能及时获得更新的服务，因此缺陷能被快速修复，缺陷修复的成本极大降低，大大提高了人们对缺陷的容忍度，降低了对质量的要求。但这可以大大降低测试的投入吗?

特别是像Facebook这样耀眼的公司，其实践似乎在支持“无需建立独立的测试团队”，让开发人员来承担越来越多的测试工作，甚至承担全部测试工作。还有人提议将测试团队的一半成员拿出来做开发，这样可以解决以前单元测试不足的问题。而测试团队的另一半被抽调到客户服务（Customer Care）团队中，负责需求前期调查和后期技术支持，在需求上加大投入，帮助建立软件产品的客户验收标准，让开发人员基于验收标准完成软件系统的设计和编程，从源头上提高质量，而且后期技术支持也得到了加强。

专业测试团队要不要？

一系列新的实践似乎在加剧质疑“专业测试团队的存在意义”。但在传统软件测试理念中，则强

调测试的独立性和专业性，专业性越强，越需要全职人员。这些实践在传统产业的质管理实践中已得到充分的验证。传统产业的质检部门是独立的，质检人员也是全职的。如果不管三七二十一，将测试团队拆分掉，会不会带来新的问题？比如：

- 究竟要不要专业的测试团队？
- 软件测试由谁来做更有效？
- 专业的测试团队未来的路会是怎样？

这一系列新的问题开始困扰业界，因此我们有必要进行充分的讨论，澄清这些问题。即使不能给出令每个人都信服的答案，也要帮助测试人员获得更客观、更理性的认识。

从软件测试本质来看，测试就是对软件产品（包括阶段性产品）质量进行全面地评估，以了解产品质量当前的状态，从而为项目管理和决策提供客观的依据。在这过程中，发现缺陷、暴露产品质量风险等，都可以看作软件测试的副产品。只是因为缺乏可操作的、具体的质量标准，所以目前没有能力和手段对软件产品做到100%的客观评价，也很难通过工具对软件产品直接进行质量指标的逐项验证而给出“质量检验通过”或“质量检验不通过”的结论。而且，软件质量整体都比较差，人们也很难通过一次性的测试完成软件产品的质量评估，而是要进行持续测试或多轮的测试，其结果弱化了“全面评估软件质量”的作用，而强调软件测试是努力发现软件产品缺陷、暴露质量风险、不断提供质量反馈的过程。

如果让构建软件产品的开发人员来评价自己的产品，那么测试结果具有说服力吗？暂且不说，王婆卖瓜，自卖自夸的嫌疑。从心理学角度分析，要发现自己的问题、否定自己的实现，需要克服心理上很大的障碍，这实际上是很不容易的。更何况人天生有惰性，希望某一个团队把事情从头到尾做好，完全依赖每个成员高度的责任感和主动性，这是不现实的。监督机制不可缺失，正如建筑需要监理、警察还有督察等，独立测试缺失，质量难以保障。在敏捷Scrum开发模型中（如图1所示），开发和测试在持续构建和持续测试之后，也依旧要设立一个独立的验收测试阶段，其实就是对独立测试的一种诉求。

基本上，大家已达成共识：单元测试、集成测试一般由开发人员做效率更高些，而系统测试和验收测试由测试人员做比较好。为什么会有这样的共识呢？

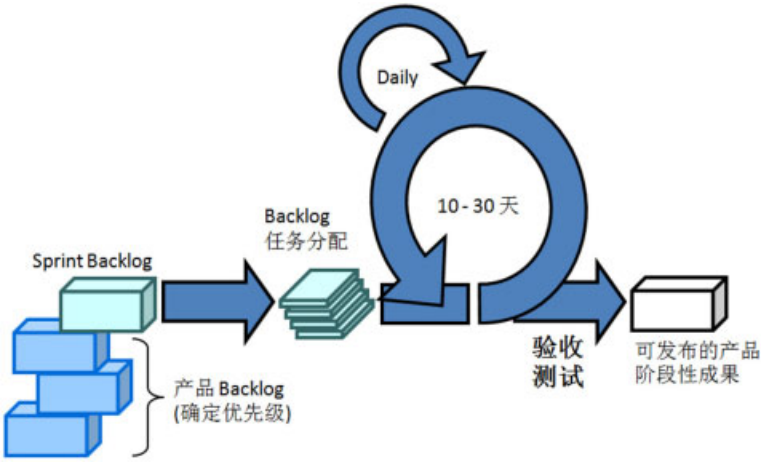


图1　敏捷Scrum开发模型示意图

如果开发人员先实现再测试，其测试的思路一定会受到实现的影响，不能达到良好的测试效果。如果先设计测试再实现产品，即采用TDD开发软件，让测试在先而不受实现的影响，效果会不错。但问题是有多少开发团队能够真正做到TDD？而且TDD对需求有更高的要求，软件产品质量的验收标准事先需要被明确定义，然后才能做到先开发测试脚本，后开发产品代码。而现实的需求又很难做到这点。

软件质量就是客户满意度。从这个角度看，测试工作更重要的任务是要在系统层面验证是否能够全面满足业务需求、是否真正满足不同用户的实际需求。而这样的测试必须要等到系统建立起来之后才能进行，如果这时让开发人员来做测试，必然会受到之前实现思维惯性的影响，效果明显降低。其次，每个开发人员通常只完成系统的很小一部分，对整体业务无法有效地把握，而且业务场景太多、繁杂，这时很考察测试人员的专业能力。只有站得高、看得远，完成业务端到端的测试，覆盖各种业务场景，才能确保系统能够正确、有效地实现整个业务流程。

如果开发人员知道某些地方很有可能存在缺陷，那么就会设法避免这些缺陷的产生。但往往开发人员并不知道自己会犯哪些错误。而如果让专业的测试人员从不同的角度、不同的思路来测试软件，测试的效果会有效得多。

更何况测试本身具有很强的专业性，包括测试建模技术、测试方法及其工具的应用，只有专业的测试人员才能很好掌握。

举一个例子，仅仅是黑盒测试方法中一项具体的测试技术——因果分析法（cause-effect analysis），美国测试专家Richard Bender差不多用其一生的时间来研究它，才将这项技术做到极致。除系统的功能测试外，做好系统的性能测试、安全性测试等就更不容易，而且随着软件技术和应用的不断发展会不断引发新的测试问题。因此，可以说这种专业的实践和积累将是一项长期的任务。

互联网的多数应用（如新浪微博、Facebook、Google搜索）属于文化娱乐服务，受缺陷的影响非常有限。例如，新浪微博的Beta版能在线运行长达三年，但银行业务系统Beta版在线运行一天都不可能。在金融、国防、航天、通信、交通控制乃至庞大的制造业生产控制等领域，无不要求零缺陷的软件产品，其独立的专业测试更是不可缺少。最近几年，各大银行不仅建立了独立的测试团队，而且测试团队还保持30%以上的发展速度。如果考虑云计算、物联网等新技术的应用，软件系统的复杂性会非线性增长，软件缺陷造成的负面影响也不能同日而语，那么在这些领域加强测试也是必然的，对专业的测试团队的需求不但不降低，反而会增强。

软件测试的未来

近几年，敏捷测试、探索式测试、精益测试、基于模型的测试等越来越受到大家的关注。《软件测试：经验与教训》一书的作者Bret Pettichord在2003年将软件测试归为四大学派（School），四年后（2007年）又增加了一个敏捷测试学派，将软件测试分为五个学派，如图2所示。

- 分析学派（Analytic School）：认为软件是逻辑性的，将测试看作计算机科学和数学的一部分，结构化测试、代码覆盖率就是其中一些典型的例子。他们认为测试工作是技术性很强的工作，侧重使用类似UML工具进行分析和建模。
- 标准学派（Standard School）：从分析学派分支出来并得到IEEE的支持，把测试看作侧重劣质成本控制并具有可重复标准的、旨在衡量项目进度的一项工作，测试是对产品需求的确认，每个需求都需要得到验证。
- 质量学派（Quality School）：软件质量需要规范，测试就是过程的质量控制、揭示项目质量风险的活动，确定开发人员是否遵守规范，测试人员扮演产品质量的守门员角色。
- 上下文驱动学派（Context-Driven School）：认为软件是人创造的，测试所发现的每一个缺陷都和相关利益者（stakeholder）密切相关；认为测试是一种有技巧的心理活动；强调人的能动性和启发式测试思维。探索性测试就是其典型代表。
- 敏捷学派（Agile School）：认为软件就是持续不断的对话，而测试就是验证开发工作是否完成，强调自动化测试。TDD是其典型代表。

标准学派和质量学派相对比较成熟，流程、过程规范等基本已建立，包括TPI、TMMi等比较成熟，虽然未来会有一些修改。而上下文驱动是比较自然的思路，其他学派也或多或少也会从上下文去考虑，也存在融合的可能性。虽然分析学派和上下文驱动学派、敏捷学派有一定对立关系，但它们相互之间又会有更多的交融，而且敏捷方法主要以实践为基础，敏捷测试不是原发性的，而是先有敏捷开发。然后人们被动地寻求测试方法和技术来适应敏捷开发。敏捷测试缺乏自己独立的理论根基，更多地依赖于上下文驱动学派的支持，包括探索式测试和自动化测试。其中自动化测试是敏捷测试主打的王牌，没有自动化测试就没有敏捷测试，而自动化测试和持续集成、持

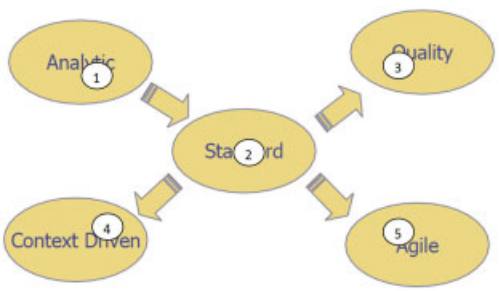


图2　软件测试五大流派示意图

续测试也相当吻合。

虽然互联网的影响越来越大，但关键系统（如银行业务、交通控制等系统）还依旧存在，关键系统会进一步促进软件开发的各种建模技术的发展，其中也包括测试建模和形式化验证。基于模型的测试也会促进自动化测试的发展，这两者之间是相辅相成的。没有测试建模的支持，自动化测试靠完全模拟手工的操作方式来实现，其实现和维护代价将相当大，使之投入产出比（ROI）总是不够理想，阻碍自动化测试的发展。当自动化测试能够借助基于模型的测试，那么自动化测试将事半功倍、如鱼得水，ROI自然也会很高。基于模型的测试，最终也需要工具的支持，例如Pairwise、因果分析法等。如果没有工具支持，测试人员就会感觉很累而不愿应用。

对软件测试影响最大的因素是软件发布模式和软件开发技术。前面已详细描述了软件发布模式，在SaaS模式中可以持续发布敏捷测试、探索式测试受到更多的关注。同时，SaaS的发展促进了各种基于云计算的服务模式诞生，软件测试的云服务模式应运而生、快速发展起来。测试公有云提供公共的、开放的测试服务，像UTest、SOASTA、SauceLab和Testin等，可以完成手机应用、Web应用或其他应用的功能测试、兼容性测试、配置测试和性能测试等。而测试私有云是某个企业为自己建立的云测试服务，将测试机器资源、测试工具等都放在云端，公司的各个团队都可以共享所有测试资源，完成从自动分配资源、自动部署到测试结果报告生成的测试过程，而且还能将测试流程、测试管理等固化在私有云内。

在软件开发技术方面，软件开发框架、工具也对测试有直接影响。例如，对分层构造软件系统而言，软件测试也可以采用分层的自动化测试技术。但未来有什么革命性的软件开发技术还难以预料，例如未来是否产生有效的开发技术能够智能地自动完成软件设计和实现的验证。但可以肯定的是，未来依靠软件生命周期的前期努力与创新构造更高质量的产品，依靠更好的单元测试技术充分实施代码层的测试，让“质量是内建的”落实到位，并借助API、UI等不同层次的自动化测试来提高测试效率。这样，软件测试投入可以越来越少，专业的测试团队规模可以不断缩小，还能保持同样的软件产品质量水平。这样，对软件企业也是好事，企业质量保障成本越低，企业获益就越大。总之，软件测试未来可能会形成两个主流方向。

- 基于模型的自动化测试：以传统测试的分析学派为基础，强调从需求分析开始，为需求或用户行为构建模型，然后基于模型自动产生和执行测试用例，它更适用于关键系统的验证。这对测试人员的技术能力有更高的要求，专业测试人员会越来越精干。如果有一天，测试团队的成员是从最优秀的开发人员中挑选，测试人员只占整个研发团队的10%左右，软件开发才到了真正成熟的时代。

- 基于云服务的测试模式：非关键系统在前期系统架构设计和代码实现上可借助良好的开发框架与工具、单元测试和持续集成等工作，在没有专职测试团队的工作情况下，也能保证产品质量处在一个基本可用的水平。然后，利用上述的公有云服务模式来完成更深度的测试，如可用性测试、配置测试、兼容性测试、性能测试都可以在云平台上自动完成。剩余的功能测试（包括业务流测试、场景测试等）就可以交给大众，通过远程服务完成测试。这些测试人员可能是业余志愿者，也可能是在家工作的专业测试人员，按任务领取报酬。

朱少民

从事软件开发、测试、QA和过程改进等工作近二十年，提倡“全过程软件测试”和“缺陷预”等先进的软件工程思想，先后出版多部著作，包括《完美测试》、《全程软件测试》等。

责任编辑：杨爽（yangshuang@csdn.net）

移动产品用户体验与设计系列（三）

精致与极致

文 / 林敏

还在联想时，特别是做乐Phone的那段日子，我几乎每周都会多次从不同的领导那里听到同一个宣示：“我们要打造极致的用户体验！”即便现在，我也经常会在一些线上线下的场合与类似的口号不期而遇。每次我都一笑而过，默默同情着为响应这个口号而努力加班的用户体验团队。因为我和我的团队都曾经历过这不可能的任务。

把用户体验做到极致意味着什么？是完美，是没有一丝一毫瑕疵的意思。市场上有哪个产品敢说自己的用户体验做到了极致？即便是苹果也不敢自诩完美，从iPhone一代的通话质量到四代的天线门，出问题的都不是可有可无的功能。没有哪家公司不希望自家体验好的，但要知道，这个世界上没有极致的体验，大家不过都是在奔向完美、极致的路上罢了。故，极致者，心向往之，仰望而不可及也。

既然不可能做到极致的用户体验，那一味强调打造极致用户体验口号自然由于僵化而失去意义。高管们呼吁极致体验本没有什么不对，但问题是许多高管除了喊口号之外，留给体验团队的只剩满头的雾水与满脸的委屈。

其实，一个好的体验团队对极致体验的追求是分内之事，那种一定要码放整齐、一定要坚持一致性原则、一定要没有错别字的偏执都是在卓越企业家和设计师身上流淌的血液。约翰•洛克菲勒、安德•格鲁夫、斯蒂夫•乔布斯，莫不如此。因此，体验团队最不需要的就是对固有追求的简单重复的口号，真正需要的是公司对当下产品的用户体验策略。换句话说，正在设计的产品在市场上打算在哪些方面实现对竞争对手的超越。这决定了用户体验在目前这个产品上的优先级是如何考虑的：是先保证待机时间还是先追求酷炫视觉效果？是先保证关键模块的体验还是先确保整体设计的一致性？是先获取用户数量还是先实现功能的完整性？是先考虑成本还是先保证原材料的可再生性？这样，团队可以把每个能够想到的地方都仔仔细细地权衡、设计、再权衡、再设计……从而产生出精致的产品。

是的，我们的目标是做出精致的产品，而不是极致的产品。精致与极致的不同大概很多高管甚至设计师都没有仔细思考过。

产品的每一次发布都应该做到精致。精致是设计思考和实现的结果，在思考和实现的过程中，团队的每一个成员都要做极致的努力。也就是说，每个个体要贡献他所能提供的最缜密的思考和最具创意的方案，然后将所有这些放在一起根据前面提到的用户体验策略来梳理做出最优的平衡设计。虽然一些个别点上的极致体验在这个阶段被妥协甚至被放弃，但那一定是出于更重要的设计思考和策略考量。整体而言，产品的设计通过这个过程实现了从个体极致到整体精致。极致思考，精致整合，这才是产品设计的正确方向。

高管们，请不要再把空洞的“打造极致的用户体验”口号挂在嘴边。请对产品策略和用户体验策略贡献你们的极致思考，并传递到用户体验团队中，打造真正精致的产品。

林敏

用户体验践行者和布道者，曾创立UU网为国内早期从业人员提供学习交流平台。现为三星中国设计研究所用户体验创新部负责人。

责任编辑：陈博（chenbo@csdn.net）

设计驱动儿童教育应用

“斑马骑士”创始人徐毅斐专访

记者 / 陈粲然

儿童教育市场的潜力与缺陷

从2005年开始，中国的第四个生育高峰来临，而这个高峰时段会持续10~15年。目前处于婴幼儿阶段的人数约有1.08亿。同时，儿童教育绝对是父母有强烈投资意愿的领域，因此作为一种刚性需求，其市场前景可期。

随着移动互联网时代的来临，一切的产品都在移动化，儿童教育产品也同样面临这一趋势。不久前，No Crusts Interactive公司总裁Carla Engelbrecht Fisher在采访中说道：“如今掀起了一股‘应用热潮’，不论是儿童还是家长都开始选择更易上手的触屏移动设备，特别是智能手机和平板电脑。”

美国风投公司KPCBCEO Rex Ishibashi表示，如今美国的儿童iPad应用市场规模超过5亿美元。他认为：“儿童被这些设备吸引，因为它们很有趣。它们符合儿童的天性。”

不仅国外的厂商在积极转型，国内的开发者也同样看到了这一趋势，就连“喜羊羊”创始人苏永乐也公开指出，如今市场生态已发生巨变，移动互联网代表着未来，他也将把精力转投AppSolution平台。

有人认为，对于儿童教育应用而言，内容是最大的门槛。但实际上，移动终端上的同类产品竞争中，设计形式的力量往往成为了决定性优势。因为儿童应用有一个很重要的特质：父母和孩子都是该款产品的用户，父母会对内容进行甄别，但儿童自身对信息的处理、理解能力相对较弱，决定他们选择的更多来自设计层面，所以为儿童设计的应用更应该注重交互设计。但目前很多儿童教育类产品还停留在传统的PC网页交互方式，未能很好地利用这些特性。

《ABCKit》等知名应用的开发者Karina Ibarra在*Designing Apps For Kids*一文中提出了儿童应用的设计要点：简短的启动画面（儿童缺乏耐心）、首页的设计（低龄儿童难以被其吸引）、应用相关设置简单（防止儿童误操作）、从“大处”着眼（易于识别）、任务机制简单（易用性原则）以及随时的赞美和奖励。

其中，他认为应用的首页对儿童很难起到作用。因为在他们的可用性测试中，孩子如果想重启或恢复一个游戏、故事或某项活动时，他们不会按返回键用来恢复游戏或返回主页，大多数孩子会直接按iPad或iPhone的Home键，先退出游戏，再找到这个图标重新启动该游戏。

国产应用依靠设计脱颖而出

在《程序员》杂志7月刊中，三星中国设计研究所用户体验创新部负责人在《做正确的加法》一文中，介绍了一款在2012年“六一”儿童节被App Store推荐的儿童应用——《鲸鱼岛的冬天》很完美地通过首页设计吸引了儿童：“他们将游戏首页加上模糊处理，使它看起来像是玻璃上结了一

《鲸鱼岛的冬天》首页上的大雾

层水雾。小朋友们都会很自然地用手去擦，从而获得清晰的页面。据说许多小朋友因为这个有趣的加法而仔仔细细地把整个页面都擦干净以后才开始游戏。产品设计，就是要这般通过细微处带来的欢乐感动用户。”

追根溯源，我找到了这款应用的开发团队——“斑马骑士”（“斑马骑士”源于“Pamakids”的音译）。这个团队的创始人徐毅斐有着非常丰富的设计与用户体验研究经验：在中科院心理所获得“工程心理学和人类因素”专业的硕士学位后，徐毅斐于2004年8月进入西门子中国研究院交互设计和技术部担任项目经理/可用性工程师，那时他工作主要面向的产品是家电、医疗仪器和自动化设备；2008年3月起，徐毅斐开始涉足移动互联网，作为中国移动通信公司研究院产业市场所的用户体验研究负责人，负责OPhone手机操作系统以及手机电视等产品的设计；整整两年后，徐毅斐于2010年3月加入创新工场，以产品经理的身份参与了《点心OS》、《布丁爱生活》以及《应用汇》等项目。

从终端到操作系统，再到移动应用，这些经验让徐毅斐对硬件和软件都有了深刻的理解。所以他在2011年8月投身移动创业时选择了一个他认为最能将这两点特性结合的领域：移动终端上的儿童教育类应用。他希望能将自己在设计、交互、用户体验上的经验与热情带到儿童教育应用市场，并弥补前文上提到的设计空白。而“斑马骑士”中其他成员的背景也涵盖了心理学、教育学、艺术设计、游戏设计、软件工程等，且大多数成员有着5年以上相关领域的经验。

徐毅斐也透露了选择儿童教育领域的另一个比较有意思的个人原因：在他的记忆中，儿时接受的都是一些令他不认同的教育方式，他不愿意将这些教育理念延续到自己的下一代，这样的情结让他希望借此摸索出一套能够让孩子快乐成长的教育方式。

在徐毅斐眼中，如果成年人互联网是一个生态圈的话，那么儿童互联网也会形成一个生态圈，但儿童生态圈更为复杂：一方面，源于这个市场所面对的用户有两个，父母跟孩子；另一方面，以前没有人给孩子做过移动互联网产品，国外也没有成熟的产品形态和成熟的模式，因此它的产品形态、商业模式都是全新的。正因为有太多不确定的东西，所以各个创业者、创业团队进来时，都会带入他们所熟悉的内容。但有一点可以确定：一切产品定位都应该从需求层面出发。

“鲸鱼岛”不代表最终模式

几乎没有任何团队能够在第一款产品就大获成功，斑马骑士也不例外。在《鲸鱼岛的冬天》之前，他们推出过一款卡片学习类应用。毕竟所有成员都是第一次进入儿童应用市场，大家也就抱着摸索方向、磨合团队的轻松心态开始了第一次练手。

这次“练手”虽然在市场上没有得到太大的反馈，但成果也颇为丰厚。他们很快发现，孩子真正需要的是一款真正能够发挥平板特性的应用，而卡片并不是一种能很好地将各种特性结合的表达方式。在传统观念里，学习与游戏的功能是截然分开的，但在观察孩子学习的过程中，他们发现自然状态下的学习就是孩子们游戏时解决问题的过程。孩子的学习就是从游戏开始的，相较于枯燥的知识灌输，游戏更容易让孩子们乐于参与其中，让他们更有动力去主动探索，更容易激发其创造力和想象力，并获得成就感，这时他们对知识的掌握也更加快速和牢固。

Karina Ibarra也在*Designing Apps For Kids*中对此作出过解释：孩子会只为了乐趣重复做上千次操做，当他们重复面对特定的刺激和信息时，就开始学习和理解。

在这种思路下，“斑马骑士”的第二款产品《鲸鱼岛的冬天》在设计模式上发生了巨大的转变：他们将天气变化等自然知识融入到游戏操作中，探索孩子在游戏中学习的方式，并将触屏手势、多点触控、重力加速度、声音控制等技术有机地加入其中。

同时，徐毅斐认为，游戏颜色是用来影响和表达情绪的，目前很多儿童游戏的颜色都以同色系为主，孩子眼前永远都是那些很粉嫩、缓和的颜色，这种影响可能对男孩子的发展起到负面影响。本着这样的想法，《鲸鱼岛的冬天》在可爱的设计模型的基础上，加入了一些对比色和加大对比度。

不过，在徐毅斐的眼里，《鲸鱼岛的冬天》也只是另一个方向的尝试而已。他最终希望打造出的儿童应用模式，还存在更多的可能。

目前斑马骑士的主要用户来自中国大陆，但他们已摩拳擦掌准备进入海外市场了。主要原因还是因为海外市场相对比较成熟，生态系统健康，只要产品好，就能够吸引用户为产品付费。而他们的第一个选择是日本的iOS市场，在开发者眼里，日本市场一直都以高ARPU值著称，但又难以进入。徐毅斐对欧美市场的产品与日本市场的产品做了对比，他认为在日本市场像《鲸鱼岛的冬天》这样风格的产品相对较少，而其欧美的画面风格设定也更符合日本用户的口味。

做符合团队特色的技术选型

在产品背后，斑马骑士也在技术上进行了大胆的尝试：他们是国内最早尝试应用Air for mobile（利用现有的Web开发技能，建立和配置跨平台的桌面应用）和Starling（Stage3D GPU加速引擎）技术的开发团队。

徐毅斐表示，这样的技术选型方案源于整个团队的背景和发展思路：“我们的技术团队原来是做网页游戏开发的，Flash产品做得较多，在技术上擅长ActionScript，因此自然会考虑把那些东西拿过来用。当时Air技术刚刚诞生，我们对它做了评估，从性能看，它经过优化，可以调用一些原生的代码，在很多方面已不输给原生应用。同时它能发挥Flash的作用，把Flash上积累的特效、效果、类库等直接使用，而在原生应用上去实现这些效果是非常废工夫的。另一方面看，它具有跨平台特性，虽然目前iPad非常流行，但它的普及率不会特别高，而Android是我们非常看重的一个市场。基于这些因素，我们从一开始就选择了Air for mobile来做《鲸鱼岛的冬天》。”

“鲸鱼岛”中采用了较大的对比度

“在做完一个Demo之后，我们向Adobe（中国）展示了这个产品，他们给推荐了Starling。Starling是一个纯粹的ActionScript 3库，模仿传统的Flash显示列表架构。和传统的显示对象不同的是，Starling的所有内容都直接提供给GPU，渲染性能较之前有很大的提升。同时，Starling是开源的，一切内容都有文档可查，能将学习成本降到最低。在这个思路下，我们把架构重新调整到Starling架构下。”

国内游戏产业现状十一谈

文 / 郑金条

本文从相对宏观的层面去窥视一些国内游戏行业的现状和趋势，将其归结为十一个元素，包括：同质化、换皮术、山寨术、数值化、微型化、页游化、广告依赖、跨平台、海外市场、免费模式和服务型游戏、监管综合症。

同质化

素材严重同质化最明显的特征就是三国题材游戏的泛滥。仅以三国再加上一些前后缀命名的游戏就数不胜数（因为文化熟知和认同度的关系，部分基于史实和演绎的大众文化在不同形式的表现下都能够获得追捧，诸如三国系列或者西游系列，仍然有大量的开发者还寄望着能够以相似的题材再切入市场以获得一定的商业份额）。而一些相似类型的游戏（题材相似、玩法相似、目标用户群重叠），诸如目前火爆的寻仙游戏、魔幻游戏或在博弈领域拥有大量爱好者的农牧场游戏、城建游戏、餐厅游戏以及捕鱼类型游戏，同质竞争充斥着玩家市场，有些开发者并不介意在单一平台上有数款与自己类型相似的游戏，诸如热酷的《梦幻海底》和《捕鱼假日》（在Qzone平台都有不俗的收益）。

同质化最潜在的风险是挑战了玩家的新鲜感，如果不能够在玩法或游戏节奏上（包括游戏画面风格以及声效）推陈出新的话，很容易失去竞争力。

换皮术

在游戏领域（特别是网页游戏），换皮的嫌疑最容易引起玩家市场的争议（无论是形式还是重新回炉打造，以经典再生的名义出现），事实上不少以续集名义出现的游戏也因为新增的游戏元素太少而有了换皮的特征，这其中包括以下几个元素。

- 成本元素。特别是在之前并未获得市场认可的情况下，开发商从成本角度考量的对之前投入的挽救方式，这种状况下再度获得市场认可的可能性并不高。
- 新渠道元素。获得新的推动渠道，面对差异化的玩家所作的元素改动，诸如一些相对显得本地化的特征。
- 流水操作惯性。游戏生产已延续为流水工业线，在基本骨架不变动的情况下，推出具有外部形态差异的新游戏，可能存在连续性的成功。

事实上，这是相当独特的山寨术，可能是某个开发者团队对自己产品的山寨，也可能是公司内部根据用户的喜好特征不断重复自己被市场印证的作品。

山寨术

山寨是国内游戏中最绕不过去的坎儿，基本沿袭了两条路线：对海外典型案例的山寨（目光大都一致紧盯着在Facebook或App Store上大放异彩的

游戏，诸如轰动一时的《Draw Something》、《Temple Run》、《Fruit Ninja》、《Cut The Rope》、《Doodle Jump》都不乏强力抄袭者，而有些山寨型团队甚至一跃成为国内顶尖的游戏研发团队，旗下拥有数以千万计的累计用户量）和对创意产品的山寨（这个在国内被诟病最多的是实力公司对创业型公司的复制，当然也包括对原游戏产品核心团队的挖角行为，而在运营力的差异下，很容易呈现出高下立判的趋势，大多无力抵御强势冲击的研发公司只能黯然接受被逆向淘汰的命运）。

大量排行前列的三国素材游戏

早先我们曾很认真权衡过山寨心态的衍化以及对大面积开发者的影响，其中包括对品牌的伤害（例如被定位为习惯性抄袭者）、养成企业的惰性并使借鉴成为一种必须的常态等，但这些显然无力抵挡任何快经济的驱动（减少研发周期和成本，并在原先市场认可的基础上侥幸瓜分一定比例的市场，以完成最原始的累积）。

数值化

在浏览器低端化（诸如IE6的高占比）以及技术革新存在障碍的现实背景下，大量的游戏（特别是网页游戏和社交游戏）采纳的是非常普通的画面呈现效果以及不算高端的游戏引擎，从而依靠数值逻辑来调动玩家参与游戏的积极性，诸如：调整经验级别以控制游戏的生命周期；通过数值调配以实现部分玩家具有超级游戏影响力；或以比值的关系让玩家参与游戏道具或者其他稀有属性产品的竞争。

特别是以其中的第二点（通过数值调配以实现部分玩家具有超级游戏影响力）在游戏中的运用更为娴熟，在有些游戏设定的VIP玩家级别中，除了相应特殊的权利外，对超额数值量的获取也是玩家追逐的主要方向（有些游戏的VIP级别划分相当细微，达到十数级，再按照每一级的属性关系给玩家获得超额数值的能力，在满足玩家追求超越和虚荣的背后，则是不同级别高额的申购投入）。

还有利用隐性概率让玩家在某些稀有品类的获取上孜孜不倦投入的博彩功能（诸如开宝箱，如果不具体指明获取概率的玩法都具有博彩属性）。尽管有政策上的顾忌，但同样挡不住开发者在网页游戏或手机游戏中以替换马甲的形式一再出现，并且成为某些游戏的主要营收来源。

微型化

在客户端游戏形势不明朗（诸如海外超级游戏的阻击、国内同类产品的强势竞争、运营渠道的单一化、运营成本高昂，以及相对极端的代理运营模式），而网页游戏的竞争又趋于惨烈（过于集中的同质化、用户获取成本高价化、游戏品质提升带来的连带研发成本压力，以及相对无序的竞争环境）的情况下，社交游戏和手机游戏以相对低门槛介入，吸引了大量开发者在看似蓝海的市场上放肆地角逐，不管是实力游戏公司对新市场的强势卡位还是新晋研发公司对新市场的放手一搏，都将游戏的微型化带向了新的规模高度，其中以Facebook、Qzone、App Store和Google Play的平台驱动最为有力。

尽管社交游戏在国内的营收困局，手机游戏的海外突围战困局都让国内的开发者在迷茫失意，但随着休闲、移动和关联三大理念在用户群和开发者层面的渗透，社交游戏和手机游戏看起来会是被定义为顺“势”产业。

网页游戏化

就国内目前的产业环境而言，网页游戏似乎必然会成为各种游戏形态的临时汇集过渡点，在网页游戏以无端游戏的名义逐步从大型网游中接过大旗，成为玩家的娱乐新选择时，社交游戏在短暂的用户兴奋期之后出现了营收彷徨，也开始将自己重新衍化为网页游戏。

在获取用户成为优势，而营收屡屡成为短板的社交游戏领域，包括顶尖的热酷游戏、恺英游戏都寄望着从网页游戏中获取更好的账面报表。

至于一向以休闲益智领衔的手机游戏也在最近呈现出了向手机网游靠拢的趋势，这点在App Store中国区的营收榜表现的更为明显，典型的诸如在网页端流行的《神仙道》、《胡莱三国》。

以《神仙道》为代表的页游在移动端也取得了不俗的成绩

广告依赖症

我们曾认为一个合理的产业链除了研发商、运营平台和用户之外，中间的环节服务商（诸如广告服、资讯服务、统计服务、托管环境服务，以及资质申请服务）将成为另外一股不可或缺的力量，而实际上，无论是网页游戏、社交游戏还是手机游戏，在游戏数量快速膨胀的情况下，中间广告行销服务商都正在崛起，诸如正当热门的Kiip或者Tapjoy之类的。

而寻常所见的游戏营销手段则不外乎是：平台的广告投入（平台的优势位置推荐、平台的价值用户导入、其他游戏的推荐导入）、搜索引擎的SEO或者SEM广告投放（基于百度和Google的搜索引擎优化以及关键词广告购买）、基于PC或者基于Mobile的广告联盟（基于各种站点的广告植入和分成植入）、典型的游戏媒介（诸如新浪游戏或者17173游戏网站）、社交网络（博客、论坛、微博以及邮箱关系链的游戏信息推送）、线下户外广告（诸如公交车身、地铁或者户外站牌广告）。当然也可能包括了一些恶意竞争，例如在SEM广告投放中对竞争对手的恶意拦截或者在游戏媒介的广告营销中使用了超过底线的广告展示形式。

事实上，大量的游戏效益都在各种广告循环中成为行销商的营收，要么处于笼络用户的广告投放，要么在游戏达到一定高度后为了维系游戏的用户群水准而不断执行广告支出。

跨平台

Matmi创始人Jeff Coghlan提出一个很典型的观点：苹果和App Store改变了整体的游戏生态环境（触屏体验、即时体验、创意主导），而接下来跨平台的游戏追求将会更具意义，这和PopCap手机业务产品和商业策划资深总监Giordano Contestabile的观点相近——他认为未来最大的趋势就是游戏的跨平台运作，特别是PC端和手机端的游戏移植；当然EA高级副总裁Andrew Wilson的论述更彻底，他认为未来收获巨大的胜者将是那些成功创造跨平台游戏体验的公司。

与之应运而生的是市场上到处鼓噪着各种跨平台工具，诸如Ansca Mobile、Sibblingz、GameSalad或以及HTML5语言，实现从PC端向移动端的衔接，包括诸如《神仙道》或者造《造梦西游3》等网页游戏，以《胡莱三国》、《开心水族箱》为代表的社交游戏，它们正在向手机端迁移，并取得了不俗表现。

Beau Hindman认为未来的游戏形态应该是这样

的：用户能够随时在有游戏意愿的情况下在能就近获取的设备上进行游戏。

海外市场

尽管海外市场是更为陌生的商业环境，而且随时面临当地的政策监管以及本土游戏的强势阻击，但这一切对于迫切需要拓展市场以获得营收空间的开发者而言并不算太严峻的障碍，特别是在目前的国内游戏商业环境下（诸如手机游戏产业的山寨丛生、极低的用户付费意愿，随处可见的刷榜作弊以及让所有人怨声载道的黑卡问题）。

除了文化理念更为相近的东南亚市场（传统国产游戏的出口目的地），随着新游戏的崛起和发行环境的改善，新区域正在形成新的发行环境，包括以社交网络为表征的区域发行市场（诸如Orkut的南美市场、Facebook的北美市场和繁体中文市场、StudiVZ的德国市场）和以App Store、Google Play为表征的全球发行市场（大量手机的发行选择都是泛全球化的，特别是以英语为首选语言的游戏）。

免费模式和服务型游戏

不管是Free To Play的免费游戏还是在Paid之后提供IAP模式的付费游戏，和完整付费游戏最大的差异是所有的玩家都不再一视同仁（诸如相似的游戏体验），取而代之的是差异化的游戏体验（诸如更可观的数值获取，更强势的道具和装备获得，更有号召力的游戏影响），从而将游戏呈现的核心视角转向两个层面：第一个是哪些环节能够触动玩家在游戏中付费；第二个是如何呈现出付费游戏玩家在游戏中的差异化体验（包括游戏荣耀）。

而这些方式的实现无外乎就是将游戏向服务型迁移，将游戏的创意产业做实为彻底的开放式产品服务，并偏向服务极少数的优势玩家以完成游戏的营收获取。从而产生了四种评判姿态：没有付费意愿，且没有可迁移景象的；没有为当款游戏付费意愿，但可实现新游戏迁移的；有付费意愿，但支付额度一般的；有强烈支付意愿，并且有可能成为游戏极少数的Whales用户。

在国内，开发者关于Whales用户的严重服务倾向是相当明显的，整体上有时候游戏显示出的就是竞争娱乐（诸如神仙道的VIP等级、小白大作战的VIP等级，基本等同于我们前文提到的数值化）。

监管综合征

国内游戏除了定向用户市场的困惑，另外一层不可避免的压力来自政策监管。

但事实上，让开发者更困惑的是监管的无序和人为弹性，以至于开发者很难甄别哪个领域是适合发挥的（例如新生的社交游戏和手机游戏，在纳入监管后，如何保障整体产业的活力）；哪些又必须在彻底的方圆规则中寻找生存的缝隙（诸如哪些题材是属于禁忌的，哪些表达形式同样是禁忌的，而哪些广告形态是不至于触碰底线的）。

事实上，更多的人并不算清楚，包括可行的通道和不可行的通道，以及在这些评判之中到底存在哪些人为弹性（比如什么样的情况下是允许的，而什么样的情况下又是不被允许的）或者不同的审核者在面对同样问题的情况下又怎样去评估相似应用审核之间的差距。

郑金条

游戏邦负责人，游戏邦主要关注和解析国内外社交游戏和手机游戏领域，并定期做深度行业阐述。

责任编辑：陈博（chenbo@csdn.net）

创建更加灵活的 App

文 / 屠毅敏

在2012年7月27日-28日由CSDN举办的iOS/Android训练营上，屠毅敏讲解了在Android和iOS上的两种基于原生应用开发的灵活性解决方案，可以在不改变开发人员技能要求和已有的开发方式的前提下，赋予应用在发布后改变程序行为的能力。本期《程序员》中，他将该主题撰写成文，分享自己的经验。

移动应用的部署方式，即发布→下载→安装→运行，决定了它不具备Web的高灵活性。这是多数互联网公司在移动应用开发的过程中所遇到的一个限制。如何打破这一瓶颈，让程序更灵活，让开发更敏捷，让迭代更迅速，是提高开发流程整体效率的一条捷径。

HTML5的发展为开发人员提供了这样一种可能，例如PhoneGap就是目前比较成熟的结合原生与HTML的一种方案。其带来的优势不仅在于跨平台性，同时它可以将业务逻辑部署在服务器上，不需要进行客户端的升级即可灵活地进行调整。

但就目前来看，HTML5要全面达到原生应用的用户体验还有很长的路要走。并且与原生结合的开发方式虽能够取长补短，但同时也会对程序的复杂度、可维护性以及开发人员的技能提出了更高的要求。

本文分别介绍了Android和iOS上的两种基于原生应用开发的灵活性解决方案，可以在不改变开发人员技能要求和已有的开发方式的前提下，赋予应用在发布后改变程序行为的能力。相关的代码及示例已共享为开源项目：http://github.com/mmin18。

Android

所有的Android应用都被封装在APK包中，一个APK包一般由三部分组成。

- AndroidManifest.xml描述了APK包的固有属性，是不可变部分。
- classes.dex包含了Java类（.class）被编译打包成了Dalvik虚拟机的中间代码。
- res/文件夹包含了所有可访问的资源文件。

一个Android应用在启动时，首先是由Dalvik加载Android自身的框架，之后会加载APK包中的classes.dex文件到全局的ClassLoader上，最后根据AndroidManifest.xml中指定的类名，创建对应的Activity实例来展示UI。

Android通过dalvik.system.DexClassLoader提供了动态加载Java代码的能力，如果能够在Activity启动之前，替换全局的ClassLoader（Application.mBase.mPackageInfo.mClassLoader），那么我们就可以改变载入的Activity类对应的实现。

ActivityLoader

ActivityLoader和ActivityDeveloper项目（https://github.com/mmin18/AndroidDynamicLoader）展示了利用Java反射机制替换全局ClassLoader到我们自定义的dalvik.system.DexClassLoader，从而在程序运行过程中改变Activity的具体实现。

ActivityDeveloper为开发环境，开发完成后，将生成的APK文件（不需要签名）放入ActivityLoader的资源文件中，并重新启动ActivityLoader来装载新的代码，就可以使得ActivityDeveloper中的代码在ActivityLoader中运行。

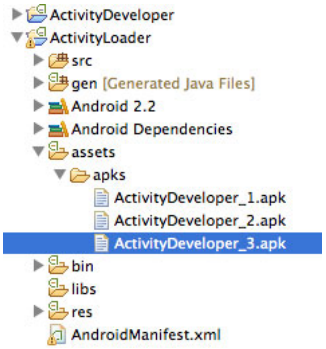


图1 ActivityLoader

但这种方式也有局限性。首先在ActivityDeveloper中开发的Activity必须和ActivityLoader.AndroidManifest.xml中指定的类名相同，并且由于AndroidManifest.xml是APK包的固有属性，无法在运行时改变，所以无法动态增加Activity。另外，我们修改的变量并不在Android的文档中提及，所以无法保证在Google升级Android系统后该方法仍然有效。

FragmentLoade3r

每一个Activity的上下文都包含了自身的ClassLoader和资源管理，通过重载以下四个函数，就可以替换Activity所载入的资源和代码的路径指向。

- ClassLoader getClassLoader()
- AssetManager getAssets()
- Resources getResources()
- Theme getTheme()

在Android 3.0引入Fragment后，我们可以在一个应用程序中只实现一个Activity作为入口和运行的容器，在启动的Intent参数中指定Activity需要动态加载的Fragment类名、classes.dex文件和资源文件路径，并在启动完成后把所有的界面响应、生命周期管理和状态可持久化等均交由Fragment处理。（Google在Android 3.0后引入了Fragment，Fragment既有Activity的生命周期管理和状态可持久化，又可以像View那样自由组合。）

FragmentLoader和FragmentDeveloper项目（链接为https://github.com/mmin18/AndroidDynamicLoader）实现了利用Fragment来负责UI和所有的代码逻辑，Activity作为唯一的启动入口，负责从APK加载Fragment及相关的资源。其项目开发在FragmentDeveloper中进行，FragmentLoader类似模拟器，从外部加载APK并运行。

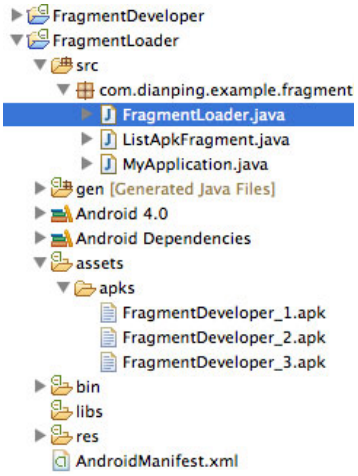


图2 FragmentLoader

利用这一开发方式，我们可以在应用部署后完全通过动态加载的方式来更新客户端运行的代码和资源文件。该方式的限制主要是无法在运行时变更AndroidManifest.xml的定义，如程序的权限、版本等。

iOS

虽然Objective-C的运行时支持新增类型和方法，但由于苹果的限制，开发者无法在iOS上动态加载Objective-C原生代码，所以只能寻求替代方案。

脚本语言就可以在一定程度上解决这一问题，Wax项目（链接为http://github.com/probablycorey/wax）实现了在运行时使用Lua创建Objective-C类与方法的能力，使得用Lua语言开发iOS应用成为一种可能。

当然，多数熟悉iOS的开发人员不一定会赞同采用这种开发方式，因此我们在引入Wax的同时希望能够和已有的iOS项目无缝地结合起来，不改变项目使用Objective-C开发的方式，而是在项目上线后通过Wax来改变程序运行时的行为。这样就可以避免App Store上漫长的审核，随时对线上程序进行调整，甚至是修复线上程序的问题或缺陷。

图3　原版与补丁版运行结果比较

WaxPatch项目（链接为：http://github.com/mmin18/WaxPatch）展示了Lua补丁的功能，在程序启动时会从指定地址下载一个包含所有Lua补丁的Zip包，通过Wax加载后改变了既有Objective-C实现方法的指向函数，从而改变了程序的行为。

图3左侧是使用Objective-C开发的原版程序，其展示了一个UITableViewController，主要代码如下：

```
@implementation MainViewController

- (NSInteger)tableView:(UITableView *)tableView
 numberOfRowsInSection:(NSInteger)section {
    return 10;
}

- (UITableViewCell *)tableView:(UITableView *)
 tableView cellForRowAtIndexPath:
 (NSIndexPath *)indexPath {
    UITableViewCell *cell =
      [[UITableViewCell alloc]
       initWithStyle:UITableViewCellStyleSubtitle
       reuseIdentifier:@"Cell"];
    cell.textLabel.text =
      [NSString stringWithFormat:@"%d",
       indexPath.row + 1];
    return cell;
}

@end
```

而图3右侧显示的是在加载了Lua补丁后，程序的运行结果。

Lua补丁覆盖了原程序的tableView:cellForRowAtIndexPath: 方法，Lua脚本如下：

```
waxClass{
  "MainViewController", UITableViewController}

function tableView_cellForRowAtIndexPath
  (self, tableView, indexPath)
    -- 被覆盖的方法会在名字前加上'ORIG'继续保留
    local cell =
      self:ORIGtableView_cellForRowAtIndexPath(
       tableView, indexPath)
    cell:textLabel():setText(
      "" .. (10 - indexPath:row()))
    cell:detailTextLabel():setText(
      "http://github.com/mmin18")
    cell:textLabel():setTextColor(
      UIColor:redColor())
    return cell
end
```

可见Lua脚本的编写方式和Objective-C非常相似，因为所有的方法和类名都和iOS的原生框架保持一致。一个iOS开发人员只需略微了解Lua的语法，就可以编写对应的Lua补丁。更多的细节可以参见Wax项目的介绍（链接为：https://github.com/probablycorey/wax/wiki/Overview）。

需要注意的是，Wax项目本身不支持方法覆盖和从Objective-C反向调用Lua修改后的实例，WaxPatch项目对Wax框架做了一定的修改，具体的实现请参见开源项目。

总结

Android框架本身就是一个插件系统，Fragment-Loader项目利用了这一特性，采用极少的代码就实现了较完善的动态加载特性，几乎所有的业务逻辑代码和界面所需的资源文件都可以实现动态加载。

在iOS下使用WaxPatch的目的是引入灵活性，在已有程序的基础上修改部分代码逻辑，避免漫长的App Store上线审核，这在程序需要紧急修复线上缺陷的情况下非常管用。

移动互联网的发展非常迅速，而Web开发中的成功模式，例如最小化原型、快速验证、灰度发布等也会越来越多地被移动应用开发所采用。这都需要程序灵活多变的特性来进行保证，希望以上介绍的项目能给应用开发者一些启发。

屠毅敏

大众点评移动应用架构师，主要负责iOS和Android的应用架构设计及新技术研发，主要研究方向还包括移动网络优化、音频、定位系统及架构易用性。

责任编辑：陈博（chenbo@csdn.net）

Android软件安全开发实践

文 / 肖梓航

Android开发是当前最火的话题之一，但很少有人讨论这个领域的安全问题。本系列将分两期，探讨Android开发中常见的安全隐患和解决方案。第一期将从数据存储、网络通信、密码和认证策略这三个角度，带你走上Android软件安全开发实践之旅。

过去两年，研究人员已发现Android上的流行软件普遍存在安全缺陷或安全漏洞。漏洞频发的原因可能有很多，例如以下几种。

- 与一切都是集中管理的iOS相比，Android提供了一种开放的环境，在获得了灵活性、可以满足各种定制需求的同时，也损失了部分安全性。
- 开发团队通常将精力集中在产品设计、功能实现、用户体验和系统效率等方面，而很少考虑安全问题。
- Android提供的安全机制比较复杂，开发者需要理解它们，并对常见的攻击思路和攻击方法有所了解，才能有效地保护软件。
- 一方面，目前很少出现对特定移动软件安全漏洞的大规模针对性攻击，在真实的攻击出现之前，许多人对此并不重视。另一方面，利用这些漏洞展开攻击并不太难，许多攻击方法和工具都已经成熟。一旦出现这种攻击，用户的个人隐私数据可能发生泄漏，账户信息可能被盗取，如果与钓鱼等攻击结合，甚至可能产生经济损失。产品开发团队则可能由此面临信任危机和法律风险。

我在此前进行的一些安全评估中，看到不少开发团队已具有非常高的安全开发水平，但也发现有知名企业的软件存在各种缺陷。在本文中，我们将向大家介绍Android软件中比较常见的安全缺陷或安全漏洞，分析产生问题的原因，介绍可能的攻击方法，并给出解决问题的建议。希望能抛砖引玉，引起大家对这类问题的关注。

数据存储

Android软件可以使用的存储区域分为外部（SD卡）和内部（NAND闪存）两种。除了大小和位置不同之外，两者在安全权限上也有很大的区别。

外部存储的文件没有读写权限的管理，所有应用软件都可以随意创建、读取、修改、删除位于外部存储中的任何文件，而仅仅需要申明READ_EXTERNAL_STORAGE和READ_EXTERNAL_STORAGE权限。

内部存储则为每个软件分配了私有区域，并有基于Linux的文件权限控制，其中每个文件的所有者ID均为Android为该软件设立的一个用户ID。通常情况下，其他软件无权读写这些文件。

关于数据存储可能出现的问题包括以下几种。

将隐私数据明文保存在外部存储

例如，聊天软件或社交软件将聊天记录、好友信息、社交信息等存储在SD卡上；备份软件将通信录、短信等备份到SD卡上等。如果这些数据是直接明文保存（包括文本格式、XML格式、SQLite数据库格式等）的，那么攻击者写的软件可以将其读取出来，并回传至指定的服务器，造成隐私信息泄露。

较好的做法是对这些数据进行加密，密码保存在内部存储，由系统托管或者由用户使用时输入。

将系统数据明文保存在外部存储

例如，备份软件和系统辅助软件可能将用户已安装的其他软件数据保存至SD卡，以便刷机或升级后进行恢复等；或者将一些系统数据缓存在SD卡上供后续使用。同样的，如果这些数据是明文保存的，恶意软件可以读取它们，有可能用于展开进一步的攻击。

将软件运行时依赖的数据保存在外部存储

如果软件将配置文件存储在SD卡上，然后在运行期间读取这些配置文件，并根据其中的数据决定如何工作，也可能产生问题。攻击者编写的软件可以修改这些配置文件，从而控制这些软件的运行。例如，如果将登录使用的服务器列表存储在SD卡中，修改后，登录连接就会被发往攻击者指定的服务器，可能导致账户泄露或会话劫持（中间人攻击）。

对这种配置文件，较安全的方法是保存到内部存储；如果必须存储到SD卡，则应该在每次使用前判断它是否被篡改，例如，与预先保存在内部的文件哈希值进行比较。

将软件安装包或者二进制代码保存在外部存储

现在很多软件都推荐用户下载并安装其他软件；用户点击后，会联网下载另一个软件的APK文件，保存到SD卡然后安装。

也有一些软件为了实现功能扩展，选择动态加载并执行二进制代码。例如，下载包含了扩展功能的DEX文件或JAR文件，保存至SD卡，然后在软件运行时，使用dalvik.system.DexClassLoader类或者java.lang.ClassLoader类加载这些文件，再通过Java反射，执行其中的代码。

如果在安装或加载前，软件没有对SD卡上的文件进行完整性验证，判断其是否可能被篡改或伪造，就可能出现安全问题。

在这里，攻击者可以使用称 为“重打包”（repackaging）的方法。目前大量Android恶意代码已采用这一技术。重打包的基本原理是，将APK文件反汇编，得到Dalvik指令的smali语法表示；然后在其中添加、修改、删除等一些指令序列，并适当改动Manifest文件；最后，将这些指令重新汇编并打包成新的APK文件，再次签名，就可以给其他手机安装了。通过重打包，攻击者可以加入恶意代码、改变软件的数据或指令，而软件原有功能和界面基本不会受到影响，用户难以察觉。

如果攻击者对软件要安装的APK文件或要加载的DEX、JAR文件重打包，植入恶意代码，或修改其原始代码；然后在SD卡上，用其替换原来的文件，或者拷贝到要执行或加载的路径，当软件没有验证这些文件的有效性时，就会运行攻击者的代码。攻击结果有很多可能，例如直接发送扣费短信，或者将用户输入的账户密码发送给指定的服务器，或者弹出钓鱼界面等。

因此，软件应该在安装或加载位于SD卡的任何文件之前，对其完整性做验证，判断其与实现保存在内部存储中的（或从服务器下载来的）哈希值是否一致。

全局可读写的内部文件

如果开发者使用openFileOutput(String name,int mode)方法创建内部文件时，将第二个参数设置为Context.MODE_WORLD_READABLE或Context.MODE_WORLD_WRITEABLE，就会让这个文件变为全局可读或全局可写的。

开发者也许是为了实现不同软件之间的数据共享，但这种方法的问题在于无法控制哪个软件可以读写，所以攻击者编写的恶意软件也拥有这一权限。

如果要跨应用共享数据，一种较好的方法是实现一个Content Provider组件，提供数据的读写接口，并为读写操作分别设置一个自定义权限。

内部敏感文件被root权限软件读写

如果攻击者的软件已获得root权限，自然可以随意读写其他软件的内部文件。这种情况并不少见。

- 大量的第三方定制ROM提供了root权限管理工

具，如果攻击者构造的软件伪造成一些功能强大的工具，可以欺骗用户授予它root权限。

- 即便手机安装的官方系统，国内用户也大多乐于解锁、刷recovery并刷入root管理工具。
- 在Android 2.2和2.3中，存在一些可以用于获取root权限的漏洞，并且对这种漏洞的利用不需要用户的确认。

因此，我们并不能假设其他软件无法获取root权限。即便是存在内部的数据，依然有被读取或修改的可能。

前面提到，重要、敏感、隐私的数据应使用内部存储，现在又遇到root后这些数据依然可能被读取的问题。我对这个问题的观点是，如果攻击者铤而走险获得root权限（被用户觉察或者被安全软件发现的风险），那理论上他已拥有了系统的完整控制权，可以直接获得联系人信息、短信记录等。此时，攻击者感兴趣的软件漏洞利用更可能是获得其他由软件管理的重要数据，例如账户密码、会话凭证、账户数据等。例如，早期Google钱包将用户的信用卡数据明文存储，攻击者获取这些数据后，可以伪装成持卡人进行进一步攻击以获得账号使用权。这种数据就是“其他由软件管理的重要数据”。

这个问题并没有通用的解决方法。开发者可能需要根据实际情况寻找方案，并在可用性与安全性之间做出恰当的选择。

网络通信

Android软件通常使用WiFi网络与服务器进行通信。WiFi并非总是可信的。例如，开放式网络或弱加密网络中，接入者可以监听网络流量；攻击者可以自己设置WiFi网络钓鱼。此外，在获得root权限后，还可以在Android系统中监听网络数据。

不加密地明文传输敏感数据

最危险的是直接使用HTTP协议登录账户或交换数据。例如，攻击者在自己设置的钓鱼网络中配置DNS服务器，将软件要连接的服务器域名解析至攻击者的另一台服务器；这台服务器就可以获得用户登录信息，或者充当客户端与原服务器的中间人，转发双方数据。

早期，国外一些著名社交网站的Android客户端的登录会话没有加密。后来出现了黑客工具FaceNiff，专门嗅探这些会话并进行劫持（它甚至支持在WEP、WPA、WPA2加密的WiFi网络上展开攻击！）。这是目前我所知的唯一一个公开攻击移动软件漏洞的案例。

这类问题的解决方法很显然——对敏感数据采用基于SSL/TLS的HTTPS进行传输。

SSL通信不检查证书有效性

在SSL/TLS通信中，客户端通过数字证书判断服务器是否可信，并采用证书中的公钥与服务器进行加密通信。

然而，有开发者在代码中不检查服务器证书的有效性，或选择接受所有的证书。例如，开发者可以自己实现一个X509TrustManager接口，将其中的checkServerTrusted()方法实现为空，即不检查服务器是否可信；或者在SSLSocketFactory的实例中，通过setHostnameVerifier(SSLSocketFactory.ALLOW_ALL_HOSTNAME_VERIFIER)，接受所有证书。做出这两种选择的可能原因是，使用了自己生成了证书后，客户端发现证书无法与系统可信根CA形成信任链，出现了CertificateException等异常。

这种做法可能导致的问题是中间人攻击。

在钓鱼WiFi网络中，同样地，攻击者可以通过设置DNS服务器使客户端与指定的服务器进行通信。攻击者在服务器上部署另一个证书，在会话建立阶段，客户端会收到这张证书。如果客户端忽略这个证书的异常，或者接受这个证书，就会成功建立会话、开始加密通信。但攻击者拥有私钥，因此可以解密得到客户端发来数据的明文。攻击者还可以模拟客户端，与真正的服务器联系，充当中间人做监听。

解决问题的一种方法是从可信CA申请一个证书。但在移动软件开发中，不推荐这种方法。除了申请证书的时间成本和经济成本外，这种验证只判断了证书是否CA可信的，并没有验证服务器本身

是否可信。例如，攻击者可以盗用其他可信证书，或者盗取CA私钥为自己颁发虚假证书，这样的攻击事件在过去两年已有多次出现。

事实上，移动软件大多只和固定的服务器通信，因此可以在代码中更精确地直接验证是否某张特定的证书，这种方法称为“证书锁定”（certificate pinning）。实现证书锁定的方法有两种：一种是前文提到的实现X509TrustManager接口，另一种则是使用KeyStore。具体可参考Android开发文档中HttpsURLConnection类的概览说明。

使用短信注册账户或接收密码

也有软件使用短信进行通信，例如自动发送短信来注册、用短信接收初始密码、用短信接收用户重置的密码等。

短信并不是一种安全的通信方式。恶意软件只要申明了SEND_SMS、RECEIVE_SMS和READ_SMS这些权限，就可以通过系统提供的API向任意号码发送任意短信、接收指定号码发来的短信并读取其内容，甚至拦截短信。这些方法已在Android恶意代码中普遍使用，甚至2011年就已出现拦截并回传短信中的网银登录验证码（mTANs）的盗号木马Zitmo。

因此，这种通过短信注册或接收密码的方法，可能引起假冒注册、恶意密码重置、密码窃取等攻击。此外，这种与手机号关联的账户还可能产生增值服务，危险更大。较好的实现方式还是走Internet。

密码和认证策略

明文存储和编码存储密码

许多软件有“记住密码”的功能。如果开发者依字面含义将密码存储到本地，可能导致泄漏。

另外，有的软件不是直接保存密码，而是用Base64、固定字节或字符串异或、ProtoBuf等方法对密码编码，然后存储在本地。这些编码也不会增加密码的安全性。采用smali、dex2jar、jd-gui、IDA Pro等工具，攻击者可以对Android软件进行反汇编和反编译。攻击者可以借此了解软件对密码的编码方法和编码参数。

较好的做法是，使用基于凭据而不是密码的协议满足这种资源持久访问的需求，例如OAuth。

对外服务的弱密码或固定密码

另一种曾引起关注的问题是，部分软件向外提供网络服务，而不使用密码或使用固定密码。例如，系统辅助软件经常在WiFi下开启FTP服务。部分软件对这个FTP服务不用密码或者用固定密码。在开放或钓鱼的WiFi网络下，攻击者也可以扫描到这个服务并直接访问。

还有弱密码的问题。例如，早期Google钱包的本地访问密码是4位数字，这个密码的SHA256值被存储在内部存储中。4位数字一共只有10000种情况，这样攻击软件即便是在手机上直接暴力破解，都可以在短时间内获得密码。

使用IMEI或IMSI作为唯一认证凭据

IMEI、IMSI是用于标识手机设备、手机卡的唯一编号。如果使用IMSI或IMEI作为用户认证的唯一凭据，可能导致假冒用户的攻击。

首先，应用要获取手机的IMEI、手机卡的IMSI并不需要特殊权限。事实上，许多第三方广告库回传它们用于用户统计。其次，得到IMEI或IMSI后，攻击者有多种方法伪造成用户与服务器进行通信。例如，将原软件重打包，使其中获取IMEI、IMSI的代码始终返回指定的值；或修改Android代码，使相关API始终返回指定的值，编译为ROM在模拟器中运行；甚至可以分析客户端与服务器的通信协议，直接模拟客户端的网络行为。

因此，若使用IMEI或IMSI作为认证的唯一凭据，攻击者可能获得服务器中的用户账户及数据。

肖梓航

网名Claud，安天实验室高级研究员，主要方向是移动反病毒和移动软件安全，发起或参与了多个移动安全开源项目。

博客：http://claudxiao.net

责任编辑：陈博（chenbo@csdn.net）

Android敏捷开发指南

文 / 王哲

本文紧密结合移动开发方法与技术，围绕Android平台的开发探讨提供更高质量移动产品的解决方案。作者中分析了移动开发中常见的问题，从两方面阐述了ThoughtWorks使用的测试开发方案和相应的架构方法与常用工具应用，并进一步阐述了为移动开发流程所提供的持续发布方案。

随着云计算、移动互联等一系列新技术概念的崛起，新一轮的IT经济正在不断扩大发展。带来无限机遇的同时，也提出了许多有别于传统开发的挑战。近几年来，我一直在尝试各种移动项目，虽然它们在应用领域、技术类型以及工作模式等方面各不相同，但我在摸索中逐渐总结出了一些比较具有共性的问题。

移动项目中的常见问题

为了实现较好的用户体验，反复的设计与验证导致产品发布时间延长。移动应用由于其多样性的应用场景，使产品设计侧重于适应不同目标的展现方式与操作习惯。设计实现的方式与使用用户群、企业服务模式、新的科技实现手段，以及各种碎片分化的目标支持设备等一系列因素密切相关。在产品实现初期，许多的内容和形式都需要在已有开发原型的基础上，进行紧密结合用户体验的测试。不少团队花了很长时间完成了目标，可测试后又要经历反复且大量的修改。这对产品的如期发布提出了巨大的挑战，发布时间也会因此一拖再拖。即便是产品磕磕绊绊地发布了，设计的改变往往也是最让人头痛的问题。

市场导向性强，业务需求变化快与缩短产品交付周期需求之间产生矛盾。经过了一系列的市场分析、产品设计、项目研发过程后，一个移动产品终于投放到了市场。但这完全不能看做是一个项目的交付完成，恰恰相反，这只是一个新阶段的开始。在残酷的市场竞争中，用户不是产品的被动消费者，而是需求的提出者，他们会在各种下载市场（例如App Store、Google Play）发出评论对应用进行评估，从而直接影响应用的市场占有率。同时随着一系列用户体验数据的收集分析和整理，业务部门的需求递增，新功能点开始一个个被搬上研发经理的台面。这给开发团队应对需求改变以及对递增的代码结构升级能力提出了更高要求。图1展示了一个产品在过去一年多时间里首页面的变化。

可以看出在项目所经历的四个比较大的阶段中，仅一个首界面的功能，也经历了从最开始的普通列表界面，到后来增加地图功能、书签的过程。在第四个阶段中，为了满足用户注册登录等需求，首页面还进行了基于左侧滑动菜单的导航转型与登录反馈等的功能扩充。每个重大阶段的转型，都来自最真实的市场评论数据，并结合Omniture（通过收集用户数据行为分析的工具）等产品的体验数据分析与业务增长需要进行开发。

为了实现更加精细的体验效果并兼容Android的各个版本（例如图1中列表和地图间的切换需要

图1　某产品四个阶段首页面对比图

通过动画三维翻转实现等），所有这一切都必须在同一个Activity内完成交互。这给大量的页面逻辑、状态和视图层级关系的升级改造带来了很大的难度。与此相对应的坏消息是，移动应用对短周期的快速发布有着强烈的需求。即使是一个好的应用，如果没有及时保持稳定频率的更新，很快就会被接踵而来的竞争者追赶，最后落到被用户遗忘的境地。从某种角度讲，除了产品新功能的推出外，应用程序的更新也具备某种广告的职能，去强化品牌在消费人群中的地位，稳定和扩大市场的占有率。

移动团队虽小，但要求更良好的产品集成性。同一个产品，一般根据支持平台的数量以及并行发布需求，配置有多个小团队和数据整合（API）团队。每个团队的开发因为进度不同和平台特点的不同，往往在整合过程中提出各自不同的集成需求（包括数据集成和逻辑集成），例如Android的内存性能不好，要求服务端的图片质量与剪裁要和iOS有所区别；再例如有时为了降低网络性能对体验的影响，会更改设计，将逻辑分散在API整合段和设备端。这就为团队间的整合埋下了风险。事实上，这在多团队的并行开发中，并不是个别现象。

多样化的设备和版本、长期的维护开发，带来快速升高的测试成本。随着开发功能的增加，页面布局、操作响应和交互处理大量逻辑模块增加，以及越来越分化的设备和系统版本，给测试工作带来了相当大的难度。在之前我做咨询时，一个项目经理告诉我，他有一个运行了一年半的项目，当时还有一周就要上线，可仍有60个Bug，5名测试工程师因为对质量没有信心而不停加班，开发也为修改一个个错误都头痛不已。

通过技术方案寻求解决途径

为了解决上述常见移动项目的问题，在项目实践中，我们试图通过合理地运用技术方案来帮助完成高质量移动软件的目标。

实现高可维护性的代码，减少代码扩展过程中的腐化和变动带来的副作用。随着功能增加，越来越多的逻辑模块被堆砌在同一个单元内，导致代码可读性下降，维护复杂度提高。这时的修改都存在破坏原有功能的潜在风险。

如果解决这两个问题，将在很大程度上提高应用在开发过程中对产品需求改变和开发周期控制的适应能力。实践中，我们使用元素组件化开发和测试驱动开发解决方案。组件化的基本目的就是将代码的可读性置于编码过程中，通过形成独立子元素组件来代理自身的功能逻辑，并以此结构化XML的布局资源，提高可读性。同时，通过测试驱动的开发方式为代码的粒度质量提供原始保障，让代码演进过程减少编码副作用破坏其他功能的现象，从而提升代码的可维护性。

通过功能自动化测试，保证开发过程中测试成本的相对稳定和质量保障。这里要分两个部分来谈。第一部分是通过提高自动化测试的比例，减少由人工重复完成的测试。在应对多平台、多版本、反复回归测试时，自动化测试对质量的保障就显得尤为重要；第二部分是由于对需求理解偏差，产生的质量问题和因此增加的返工。这里就需要引入基于业务行为驱动开发（BDD）的自动化测试管理。让需求成为可验证的执行代码，将会巨大限度的缩小业务需求、开发和测试之间的鸿沟。

当然，我在从事多个项目的开发咨询的过程中，也曾遇到大量的需求变更，导致自动化测试废弃，从而提高成本的案例。这时往往要注意协调自动化测试金字塔，即单元测试、功能测试、UI界面测试等几个部分的比例关系，实现质量与成本的平衡。

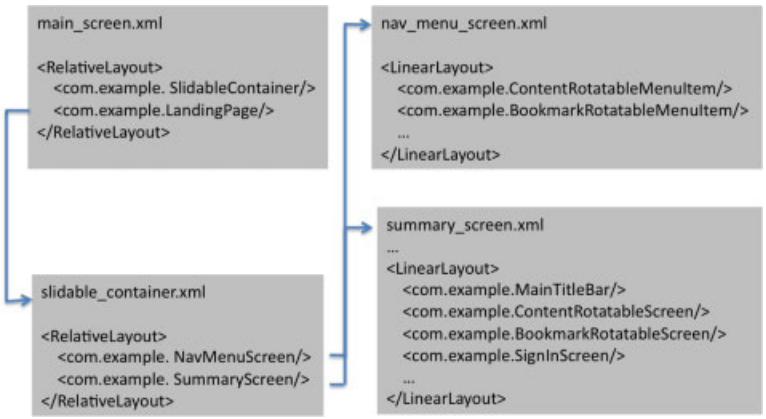


图2　某产品四个阶段首页面的组成方式

让随时可工作的产品来提高团队的交付能力。面对不断变更的需求与任何时候都可能出现的产品延时，提高团队的整体交付能力，就显得格外的重要。作为重要一环的持续集成和可用多点环境下测试，便成为这其中不可或缺的重中之重。而如果产品的各个平台都可以保证相对稳定的持续集成与发布，那么这样的方案也自然成为消除团队合作壁垒的重要技术保障。

加快用户体验验证周期，增加修改频率降低单次修改的调整规模。优秀的用户体验，永远是前端工程师最关心的部分。这就需要能够尽早地去做更深入的分析与验证，更快地发现问题，并接进行修改与调整。同时缩短反馈回到开发端的周期，将大大降低代码的修改难度，提高产品效率。而自动化的编译、集成与部署操作流程就是用一个非常有效的方法来完成闭环回路。可持续集成与部署技术为缩短周期提供了根本上的保证。

刚刚提到了许多用来移动项目问题的解决方案。那么下面就让我们来看看在Android开发领域，这些方案是如何具体实施的。

结构化组件

所谓组件化就是将应用内部的UI元素充分拆分成相互独立子部件（例如常见的Android的Widget组件）。大的子部件由多个小的子部件组成。

这样的好处是除了在代码层面上更易于修改外，同时也通过对类和方法的命名实现了XML代码的结构化。图2作为一个简单的示例，大致表现了图1中第四个阶段首页面的组成方式。可以看出XML代码中的每一个子组件都是一个相应的视图组件，这些组件通过Java类和XML的对应完成一个子视图功能。XML中大体可以表达出页面视图的一个可读性组成方法，而对应的Java文件则代表了每个视图的生成细节模型和交互响应逻辑。

另外需要说明的是，我们曾尝试了多种消息传递机制。最后证明，组件的单一顺序传递是一种相对最稳定和易理解的传递方法，子视图的交互消息只能传递给其父视图，然后由父视图传递给其他子视图。即如图2中ContentRotableScreen接收到一个自己不能处理的响应事件，应该将消息传达给SlidableContainer，再统一分发给需要处理事件的NavMenuScreen作相应的动作。

这样做既保证了消息传递的准确性，又维护了一个统一的代码结构，方便实现代码的可读性和可维护性。

单元测试工具

单元测试是测试驱动开发的主体测试构成，旨在从代码粒度上实现对应用质量的把握，也是可维护性代码的核心。其具体粒度大小取决于在代码出现问题后，能够在多大程度上准确的定位问题。这也是单元测试最大的意义所在。这份意义所带来的是更高的代码可维护性、更稳定的代码可重构性、更便捷的可扩展性。而这一切为稳定结构变化、减弱代码腐化影响、技术改进所带来的代码变更奠定了良好的基础。

为了实现比较良好的单元测试，需要一系列代码结构优化和测试工具使用的辅助。

在做深入说明Android系统开发结构之前，先让我们来看看在该平台开发时所常见的工具和相应的优缺点对比，如表1所示。

根据现有经验，我们曾尝试了表1中四种单元测试方法。很难说哪一个方案是最好的，在目前的项目实践中，我们联合应用Robolectric和Java JUnit，为了避免Robolectric速度、模拟功能不全和质量检测工具等问题，需要对Robolectric的代

码进行修改，并尽量减少对其的应用。转而通过一些结构上的应用，大量采用Java JUnit测试。

- JUnit。鼎鼎大名的Java测试框架，无数应运而生的mock框架支持，使其无论是可用性，还是易用性方面在整个Java领域无人能及。利用JUnit可以实现非常快捷单元测试。也是最常见的一种单元测试形式。

- Robolectric。这是由Pivotal Labs开发的一套开源的Android单元测试框架。其通过一系列对底层Android元素的替换来实现对原有元素调用的模拟，从而实现脱离模拟器的测试。非常值得一提的是，在测试服务器请求时，Robolectric的数据模拟和延时发送模拟，给多线程状态下的测试提供了很好的解决方法。

- Robotium。因为其对整体应用的黑盒操作特性，绝大多数技术文章将其作为功能测试工具，因此后文在叙述功能测试时也有提及。但因为其已有代码库，进行测试前需要组合编译，而且可以完成方法级别的功能测试，因此本文还是将其描述重点在单元测试时和其他工具进行对比说明。但并不等于它就是单元测试。

- Android JUnit。这是一种最常见的单元级别测试。它由Android官方提供，通过虚拟机自身提供的测试接口完成。图3源于Android开发者官网，基本上阐述了整个测试框架各个组成部分。其中，最下面方框中描述的即为框架中基于JUnit的测试部分。可以看到，其通过Android的内部测试包，调用测试执行模块完成对目标应用的测试。

该测试最大的好处是其与Android系统结合紧密，贴近真实环境。但其弊端也正是因为使用大量基于平台的虚拟，导致测试运行速度相对偏慢，影响测试效率。又因为开发过程中，单元测试是最大，也是最常规的测试，所以这种影响带来的效率降低就显得特别严重。

表1　各测试工具优缺点比较

	优点	缺点
Android JUnit	可以使用巨大部分Android控件和有比较好的设备层支持	速度非常慢、依赖平台、需要模拟器支持
Java JUnit	速度快、支持测试覆盖率等代码质量检测工具	无法做与Android UI相关的操作、与原生Java有差异
Robotium	紧密结合产品代码测试，适合完成Activity粒度的真实场景下的单元测试	依赖Android平台、速度非常慢
Robolectric	支持对Android平台依赖类底层的引用和模拟（支持有限）	无法完成UI Layout的测试，启动速度相对较慢，不利于单独Unit Test的开发

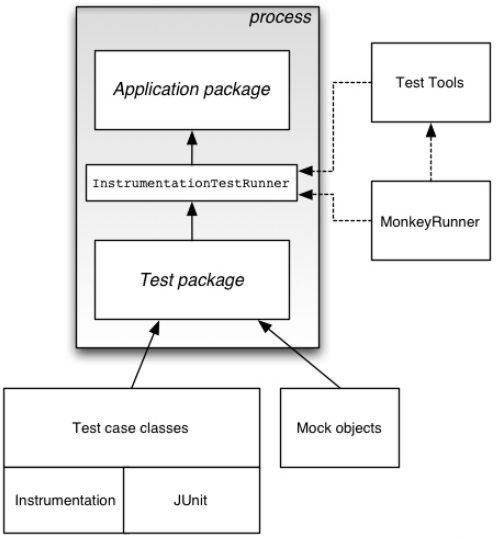


图3　Android JUnit测试框架

小结

本文我们介绍了移动开发中的常见问题、技术解决方案的基本思路，以及在具体实现中所要涉及的结构化组件和单元测试工作。在下期《程序员》中，我将继续讲解技术方案的具体实现方法，包括如何通过框架选取实现测试驱动方案，业务行为驱动的功能测试方案、持续集成、部署，以及如何让调试可持续化。

王哲

ThoughtWorks咨询师。曾多次带领团队从事完成Android、iOS、BlackBerry、Mobile Web、数据整合API等项目。致力于将敏捷先进的技术管理经验与移动产品紧密结合。

责任编辑：陈博（chenbo@csdn.net）

Cobar的架构与实践

文 / 邱硕

Cobar是提供分布式数据库服务的中间件，是阿里巴巴B2B前台应用访问数据库的统一入口。本文讲述的就是Cobar的架构演变与应用实践。

2008年，阿里巴巴B2B成立了平台技术部，为各个业务部门的产品提供底层的基础平台。这些平台涵盖Web框架、消息通信、分布式服务、数据库中间件等多个领域的产品。它们有的源于各个产品线在长期开发过程中沉淀出来的公共框架和系统，有的源于对现有产品和运维过程中新需求的发现。数据库相关的平台就是其中之一，主要解决以下三方面的问题。

- 为海量前台数据提供高性能、大容量、高可用性的访问。
- 为数据变更的消费提供准实时的保障。
- 高效的异地数据同步。

应用层通过Cobar访问数据库（如图1所示）。对数据库的访问分为读操作（select)和写操作(update、insert和delete)。写操作会在数据库上产生变更记录，MySQL的变更记录叫binlog，Oracle的变更记录叫redolog。图1中的Erosa产品解析这些变更记录，并以统一的格式缓存至Eromanga中，后者负责管理变更数据的生产者（Erosa）和消费者之间的关系。负责跨机房数据库同步的Otter是这些变更数据的消费者之一。

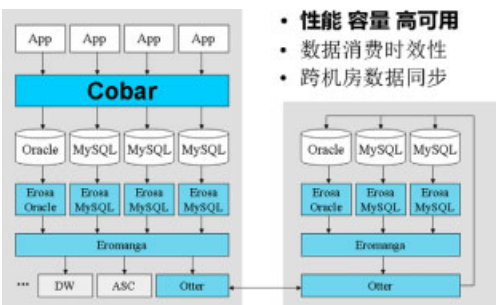


图1　Cobar是提供分布式数据库服务的中间件

截至2012年6月，Cobar的使用方已涵盖B2B中文站、国际站、国际交易、搜索、数据仓库、ITU等部门的200多个应用，其中包括中文站的Offer和国际站产品等的核心应用。

中文站Offer表的拆分和amoeba

2008年，阿里巴巴的业务基本上使用同一个Oracle+小型机做数据库，一个单点承载了多个核心业务，比如中文站的Offer应用，其数据量和访问量都很高。截至2008年，数据库的Offer表已经有1亿的数据量。在高峰时期，Oracle小型机常常load高达30，CPU利用率在90%以上。

同时，过多的应用使用同一个单点数据库，导致连接数过多，影响性能。Oracle的Standby切换在实际使用中也出现过问题，导致网站不可用这样的事故。Oracle数据库不开源，基本可以算是一个黑盒，出现的问题很难定位。随着业务量的迅速增长（2008年Offer表有1亿数据，2010年达到了3亿数量），单点的Oracle+小型机出现了瓶颈，不能满足业务需求。摆在面前的有三种选择：升级小型机的硬件；购买多台小型机以分散各个业务；以廉价PC服务器+开源数据库替代之。我们选择了第三条路，主要解决当时单点Oracle四个方面的问题。

- 数据和访问从集中式改变为分布式。

- 提供数据节点的failover。
- 解决连接数过大的问题。
- 对业务代码侵入性少。

数据分布

为了让单点数据分散到分布式的PC服务器上，基于数据表的水平拆分从一开始便被引入，成为系统的核心功能。如图2所示，MEMBER_ID字段是数据库表的拆分字段，对于某一行记录，路由算法根据拆分字段的值决定将其分布到哪个分库。

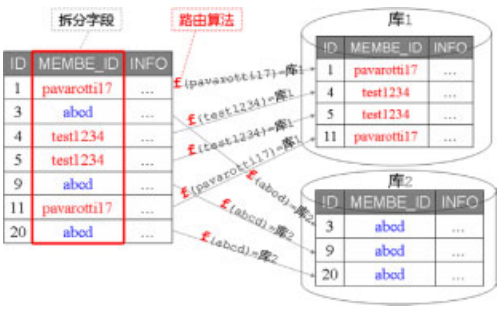


图2 数据库的水平拆分

图3描述了路由算法的过程，它使用了一致性Hash，使得扩容时数据移动较少且仅在局部移动。图3中的各个分库数据空间均匀分布（都是256），当然也是可以非均匀的，以适应不同机器的处理能力。这种水平拆分解决了第一个问题。

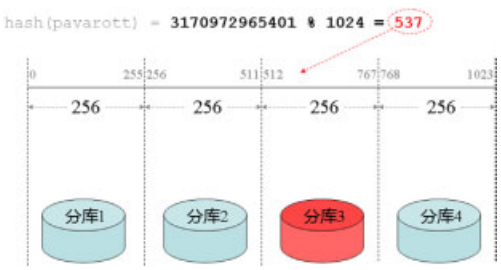


图3 路由算法的过程

第一版的系统结构

以数据水平拆分为基础，我们开发了Cobar的前身——Amoeba（下文仍旧称之为Cobar），它的结构如图4所示。

从图4中可以看出Amoeba是一个独立的进程，与应用之间通过MySQL协议进行交互，是一个proxy的结构，对外暴露jdbc:mysql://CobarIP:port/schema。对于应用而言，连接Cobar和连接

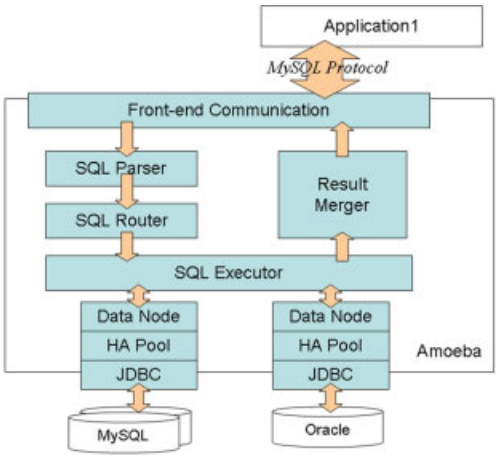


图4 Cobar的前身Amoeba

MySQL没有区别。这样做有以下几个好处。

- 无需应用引入新的jar包，从访问Oracle迁到访问Cobar可以复用原有的基于JDBC的DAO。
- 除了应用代码之外，通过普通的MySQL Command工具或者其他第三方数据库管理工具，可以像访问原有单点数据库一样访问Cobar，方便数据库的管理和问题的排查、调试。
- 数据库连接复用。Cobar使用连接池与后台真实数据库进行交互。由于Cobar的实例数量远远小于应用的实例数量，可极大程度地节约后台数据库连接（实际中，根据应用的不同，使用proxy结构后数据库连接数能够节约2~10倍不等）。虽然应用与Cobar之间的连接数和原有Oracle单点连接数相同，但Cobar本身的连接是轻量的，能够承受连接数上限远远高于后台数据库。

数据访问

Cobar通过SQL语句转发的方式实现数据访问。如图5所示，用户发来的SQL语句，Cobar解析其内容，判断该语句所涉及的数据分布在哪个分库上，再将语句转发给此分库执行（在这一版本的Cobar中，当SQL语句中涉及的拆分字段有多值，

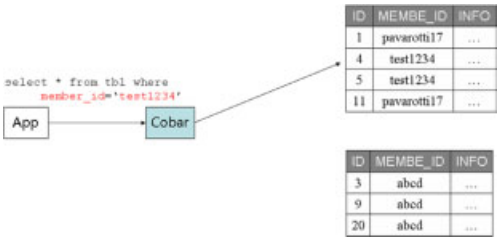


图5 通过SQL语句转发的方式实现数据库访问

如member_id IN (…)，或where条件中没有出现拆分字段时，该语句将会转发至后台所有分库执行），再将执行结果以MySQL协议包的形式送回应用端。这个过程会涉及Cobar的所有关键模块。

首先是通信模块，它负责从连续的网络数据流中识别出一个个MySQL协议包，再解析协议包识别出SQL语句输出给Parser模块。同时，把Result Merge模块输入的执行结果，编码成MySQL的协议包。它以NIO方式实现，有很高的执行效率。

这一版本的Parser负责识别出SQL语句WHERE条件中的等式（如Column1=123），以Map的形式和SQL涉及的表名一起，输出给路由模块。此时的Parser基于JavaCC实现，由文法规则自动生成（后续版本重新手写实现），文法规则涵盖了Oracle和MySQL的常用语法。

Router模块从Parser模块输出的Map中找出相应表的拆分字段的值，计算出该SQL分发到哪个分库，交由Executor负责执行。

数据源层次

水平拆分后，后台有多个数据源，对它们的管理分为两个层次，第一层是DataNode，第2层是replica。

DataNode管理拆分，一个DataNode存放一个分片的数据，彼此无数据交集。每个分片的数据存多份以保证高可用，每一份叫做一个replica，由HA层管理。每个replica表示一个具体的数据源，它是一个连接池，池内管理每一个具体的JDBC连接。路由运算只关注到DataNode层，之下的层次对其不可见。

每一份replica之间的数据复制和同步由MySQL本身的replication协议完成，同一时刻只有一个replica提供服务（称为Master，其余replica称为Slave），

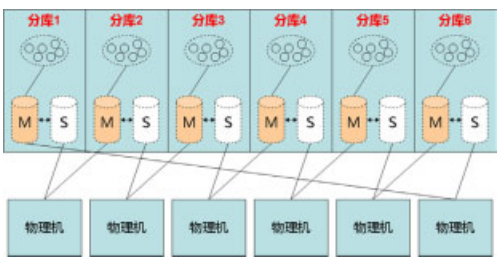


图6　物理机的部署方式

Cobar会与之保持心跳，一旦发现它不可用，会切换至另一个replica，解决Oracle单点的第二个问题。由于Slave并不对外提供服务，所以为了节约数据库的机器数量，可以采用图6中的方式部署。

MySQL采用异步的复制机制，因此多份replica间的读写分离会造成一定程度的数据不一致，因此没有被Cobar采用。

应用

这一版Cobar在2009年底开发完成，首先在规模较小的几个应用上试用。2010年，在中文站Offer应用上线。2011年下半年，包括国际站产品在内的20多个应用开始使用。这些应用原先运行在Oracle单点上，迁移至Cobar之后后端用多台MySQL提供服务，对它们进行数据迁移的过程，不能影响这些应用的在线使用，即需要进行在线迁移。在这20多个应用的实施过程中，逐步形成了一套统一的数据迁移策略，需要保证以下几点。

- 迁移过程中应用不停止或暂停服务。
- 迁移后数据一致。
- 迁移步骤中出现错误可回滚。

迁移过程涉及三个部分：应用的DAO（Data Access Objects）、Cobar和迁移任务进程，如图7所示。迁移以表为单位，过程有以下三个阶段。

阶段1：在应用的DAO代码中增加Cobar的连接，但应用的读写都在原有的Oracle上。

阶段2：应用读写仍旧在Oracle上，同时写一份数据到MySQL中。在此阶段，后台的迁移任务启动，将Oracle上相应表的全部记录读取并按照拆分规则写入MySQL实例，该过程中，前台DAO仍

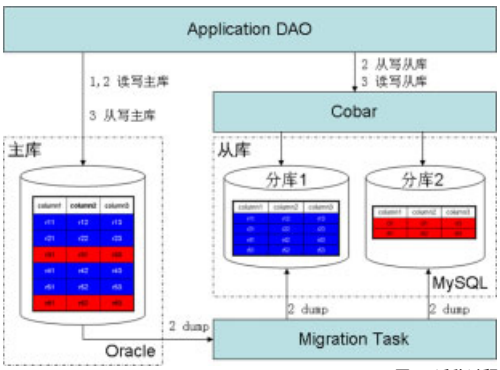


图7　迁移过程

然会对记录进行从写，可能更改或删除任务刚刚写入的数据，此时迁移任务会处理数据冲突相关逻辑。因为此时MySQL数据尚未对外提供服务，如果该阶段出现问题，可删除MySQL数据，重启一次本过程。

阶段3：阶段2结束后，主从两库的数据一致，此时，可将应用的读写迁移至Cobar，同时也从写一份数据到Oracle以便该阶段失败时回滚。

中间件的集群化

截至2011年，Cobar中间件为每个应用单独部署进程以保证应用间的隔离性，但随着应用数量的增多，Cobar服务器本身的数量也在上升。我们希望能够有一套Cobar集群服务于同一个地区的所有应用，做到跨应用的共享。为此，我们重写了Cobar，在提升性能的同时也增加一些功能。

通信模块

尽管第一版的NIO具有很高的资源利用率，但仍存在一定的优化空间，我们引入了一个ByteBuffer池，将NIO的Buffer统一管理起来，减少了NIO数据交互时的垃圾回收。重写之后的通信层，使用普通的PC服务器，Cobar实例只使用2GB的内存，单实例可以达到15万QPS。

后端去除JDBC

Cobar前端使用的是优化后的NIO通信模块，为了让该模块在后端使用，我们去除了JDBC。与后端数据库交互，Cobar直接面向协议，目前实现了基于MySQL协议的后端交互。

SQL解析和路由

原有的SQL解析器是基于JavaCC自动生成的，我们用手写的方式以LL(2)重新实现，支持MySQL 5.5的DML语法，且对它的函数和特殊语法元素，都做了很好的支持，在解析功能上做到与MySQL一致。性能比原有版本提高了10倍，解析类似这样的语句（seLEcT id, member_id, image_path, image_size, STATUS, gmt_modified from wp_image wheRe id = ? AND member_id=-123.456）只需4微秒。路由方面，实现按照多维度对表进行水平拆分，以减少语句分发的次数。

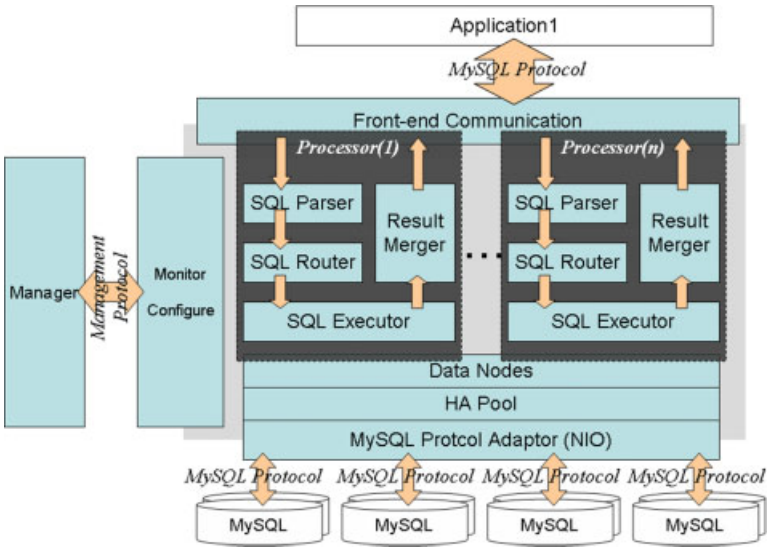


图8 最终架构

最终架构

图8是Cobar最终的架构。与第一版相比，增加了管理和监控Cobar服务器状态的Manager，同时解析路由和执行等操作被分在了单独的Processor中，以保证一定程度的隔离性。此时的Cobar只需要3-5台PC服务器，即可支撑一个机房全部应用的压力。其中一台Cobar进程的数据如下：平均QPS是3000，进程运行了9天后，只有90次FGC，而Cobar的Java进程堆内存仅1.2GB。

这一版本在2011年10月发布，至今已有200余个应用运行在它之上。

2012年6月，Cobar开源，地址为http://code.alibabatech.com/wiki/display/Cobar。7月，阿里云正在开发的火焰山项目（分布式关系型数据库的云服务）选择Cobar作为其底层中间件，实现数据分片和Proxy的需求。

邱硕

2009年加入阿里巴巴平台技术部，参与过分布式服务框架等平台产品的开发，2010年至今在Cobar团队开发分布式数据库中间件。

责任编辑：杨爽（yangshuang@csdn.net）

解析个人云存储Open API

文 / 李崇欣，蒋炜航

开放与合作是当今互联网的主题之一，而提供Open API也成为当下互联网服务提供商的一个趋势。本文从设计云存储Open API需要考虑的7个元素出发，全面解析个人云存储Open API。

个人云存储服务是这几年来日益成熟的一类云计算服务。特别是同步网盘和同步云笔记这两类云存储服务，通过同步机制为个人用户提供了跨平台、跨设备的云存储服务，成为时下流行的个人云存储服务形态。由于这两类云存储服务所提供的Open API有众多相同之处，本文将以同步网盘（Dropbox、Google Drive）和同步云笔记（有道云笔记、Evernote）的Open API为例，来讨论设计一套个人云存储的Open API需要考虑哪些因素，以及第三方应用应该如何使用个人云存储Open API。

如何设计云存储Open API

与其他云服务的Open API一样，云存储Open API的使用场景包含了服务提供商、第三方应用和用户三方。如图1所示，云存储服务提供商通过开放Open API，与第三方应用一起为用户提供更加丰富的功能和体验，提高用户的忠诚度和活跃度。

从第三方应用的角度来看，Open API可帮助其将存储服务委派给云存储服务商，从而降低自身的运营成本，专注于提供更有特色的服务。同时第三方应用还可以通过Open API接入云存储服务商的用户群，从而获得高速增长的机会。通过Open API，用户既能够从值得信赖的云存储服务提供商那里获得稳定、安全的存储服务，又能够通过授权从丰富多样的第三方应用那儿挑选适合自己需求的功能。可以看出，通过一套设计合理的云存储Open API机制，云存储服务提供商、第三方应用和用户可以达到三方共赢的局面。

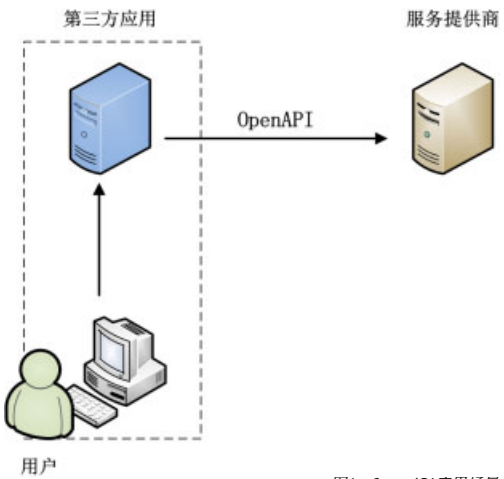


图1 Open API应用场景

设计一套云存储Open API需要考虑7个元素：授权、跨平台、向前兼容、数据操作、数据模型、数据传输效率和数据访问权限。接下来我们将结合实例来探讨这些元素。

授权

云存储服务商通过提供稳定、保密和快速的服务获取用户的信赖，成为用户的个人数据仓库。而用户名和密码就是打开这个重要仓库的钥匙。出于安全的考虑，用户不应该将自己在云存储服务商处设置的用户名和密码告诉第三方应用。OAuth就是为解决这类问题（第三方无需使用用户名和密码就能申请获得该用户授权）而设计的一套安全、开放、简易的协议。目前，绝大多数同步云笔记和同步网盘的Open API都使用了OAuth协议。

为了使用OAuth协议，第三方应用首先需要从服务提供商处申请Consumer Key和Consumer Secret来对授权请求进行签名。服务提供商通过Consumer Key可以知道Open API请求来自于哪个第三方应用，以便于对请求进行跟踪和监控。一个典型的OAuth认证授权过程如图2所示。

- 当用户尝试通过第三方应用访问数据时，第三方用户首先向服务提供商请求一个Request Token，在这个请求中，第三方应用需要带上自己的Consumer Key并通过Consumer Secret进行签名以便服务提供商验证请求合法性。
- 服务提供商在验证第三方应用后返回Request Token。
- 在得到Request Token后，第三方应用将用户重定向至服务提供商的登录页面，重定向请求中需要加入Request Token以表明身份，同时还会加入一个Callback URL参数，以便用户登录授权结束后返回该页面。
- 用户在服务提供商登录页面上进行认证授权，如果成功，则重定向至第三方应用传过来的Callback URL。
- 第三方应用被Callback后，便可以向服务提供商请求Access Toke，在该请求中，第三方应用将使用刚刚得到的Request Secret进行签名。
- 服务提供商在验证Request Token/Secret后向第三方应用返回Access Token/Secret，此时第三方应用便完成了一个用户的授权。
- 在获得Access Token/Secret后，第三方应用便可使用Access Token/Secret访问这个用户的数据。

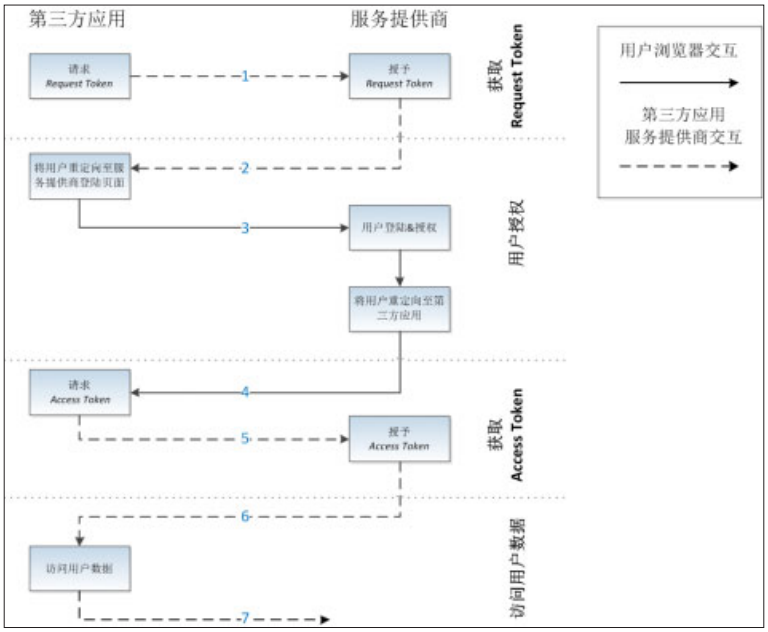


图2 授权认证流程

虽然整个流程描述起来比较复杂，但由于这个流程对于任何一个第三方应用都是一致的，所以第三方应用可以很容易地通过对范例程序的修改来实现这个认证流程。

跨平台

作为提供给第三方应用程序使用的服务接口，Open API必须有跨平台的能力。以Dropbox为例，使用Dropbox Open API的第三方客户端应用所基于的平台包括Windows、Mac、Linux、iOS、Android、Windows Phone等。同样，基于其Open API的Web应用所使用的编程语言可能是Java、PHP、.NET等。因此如何设计一套跨平台、跨编程语言的Open API是每个云存储服务商要考虑的问题。

目前大多数的Open API采用了基于HTTP的实现方式，同时也解决了跨平台的问题。为了方便第三方应用开发，服务提供商有时也针对在不同平台对HTTP进行相应的包装，提供对应平台的SDK。

另一种实现Open API的方式是提供RPC接口。相比于HTTP接口，RPC接口更便于使用者开发，使用者不必关心参数的传递方式和返回结果的解

析，就像使用本地函数编程一样。但这类实现方式的代价则是难以跨平台。由Facebook开发并开源的跨语言服务开发框架——Thrift，可以在一定程度上解决这个问题。服务提供商可以用Thrift格式的脚本来定义RPC的参数和返回结果。使用Thrift自带的编译器，Thrift格式的脚本能够被自动编码成为不同语言的实现，例如C++、Java、Python、PHP等。Evernote就是使用Thrift对外提供跨平台的Open API。

向前兼容

在发布了Open API后，服务提供商一般会定期更新Open API，提供更多功能来完善接口。但第三方往往无法及时更新自己的应用来使用最新的接口。因此服务提供商需要在一段时间内向前兼容旧版本的Open API。

基于HTTP实现的Open API，常通过在URL中添加版本号来做到向前兼容。以Google Drive为例，下载一个指定文件的HTTP接口为：https://www.googleapis.com/drive/v1/files/{id}，其中v1用来表示该API的版本信息，不同版本的API接口使用的URL不同，这样使得Open API接口也在一定程度上保持了向前兼容。

相对来说，基于RPC实现的Open API更难处理向前兼容的问题。在此，Thrift框架再次提供了一定的便利性。在保证第三方应用不改变对Thrift RPC调用方式的前提下，服务提供商可以使用Thrift编码添加新的接口和参数。当然，这需要服务提供商对Thrift的工作原理有深入的认识。

数据操作

云存储Open API为第三方应用提供的主要功能是对数据的操作。最常见的数据操作包括对数据的读取、新建、修改、删除和移动。除了这些基本操作之外，还有一些与数据相关的高级操作。例如，版本接口允许第三方应用读取用户旧版本的数据；搜索接口允许第三方应用通过关键词来定位用户的数据；分享接口允许第三方应用将用户的数据分享给其他用户。这些数据操作为第三方应用提供了灵活性和便利性。

数据模型

数据模型定义了第三方应用以何种模式来访问云存储中的数据。云网盘的数据模型往往由文件和目录组成。在这种数据模型下，第三方应用可以对云存储进行诸如创建新文件、修改已有文件的内容、创建新目录、修改目录名称等一系列操作。

云笔记的数据模型往往由笔记本、笔记以及资源组成。云笔记的笔记本和笔记可以类比云网盘的目录和文件。但与没有格式要求的文件相比，由于笔记要能够被云笔记的客户端所解析和渲染给用户，因此笔记往往采用类HTML的标记语言来描述。如果第三方应用需要保存的内容不适合使用标记语言来描述，那么它们可以使用资源。资源和文件一样以二进制数据的形式保存。云笔记中的资源往往依附某一篇笔记，而一篇笔记可以有多个资源。

除了与用户数据相关的数据模型之外，有时有些云服务也提供用户账号的访问。通过访问用户账号，第三方应用可以获得用户邮箱、云存储服务总空间、云存储服务已使用空间、昵称、注册时间等关于用户的信息，并提供更加个性化的服务。

数据传输效率

对基于HTTP的Open API来说，出于跨平台的考虑，数据结构或者对象通常以文本的形式返回给调用者，因此为了能够对数据结构或对象进行复原，通常使用XML这样的结构化描述语言对数据进行描述，在各个平台上都具有较好的通用性且易于扩展。作为一种较新的结构化描述语言，JSON在支持结构化描述时所引入的额外信息较少，因此相比于XML具有较高的传输效率，在最近几年发布的Open API中得到了广泛的应用。而对于文件和资源这样大块的数据，则通过二进制流的方式高速传输。

对于基于RPC的Open API来说，通常直接将数据结构或者对象串行化后进行传输，相比于XML、JSON这类结构化的描述语言，串行化几乎完全不需要任何额外的信息来辅助传输，同时反串行化相比于解析结构化描述语言也具有更快的速度，因此具有更好的传输效率；另外，在兼容性上串

行化通常使用直接加入版本信息的方式，这样在反串行化时便可以根据版本信息进行适当的处理。与RPC一样，串行化也存在通用性较差的问题，难以在各个平台及编程语言下通用。目前比较常见的串行化方法有Google的Protocol Buffers、Thrift Serialization及Apache的Avro等。

数据访问权限

最基础的访问权限来自于授权机制。第三方应用只有在获得了用户的授权之后才能访问该用户的数据，并且用户可以通过撤销授权来阻止第三方应用继续访问自己的数据。

除了最基础的访问权限，不同的云存储服务提供商采取不同的策略。例如Dropbox按照应用来区隔空间——第三方应用只能访问其对应目录下的文件，而不能访问用户其他文件。

Google Drive使用了更为复杂的文件级别访问权限控制。每一个文件都有对应的访问权限列表，只有当用户使用某个第三方应用开启或者新建的文件，该第三方应用才有权限访问该文件。

常见云存储Open API对比

表1是几种常见的云存储Open API的对比。但请注意：表1的信息仅供参考，最新资料请查询各个云存储服务提供商的官方网站。

表1　常见云存储Open API的对比

	Google Drive	有道云笔记	Dropbox	Evernote
授权	OAuth	OAuth	OAuth	OAuth
跨平台	HTTP	HTTP	HTTP	Thrift（limited）
向前兼容	URL 版本号	URL 版本号	URL版本号	Thrift（添加额外字段）
数据操作	读取、插入、补丁、更新	读取、新建、修改、删除、移动	读取、新建、版本、搜索、分享	新建、修改、删除、移动、搜索、分享
数据模型	文件	账户、笔记本、笔记、资源	文件、目录	账户、笔记本、笔记、资源、标签、保存的搜索
数据传输效率	JSON +二进制流	JSON+二进制流	JSON + 二进制流	Thrift Serialization
数据权限	File-level	全权限	App-level	全权限

第三方应用开发

运用云存储服务提供商的Open API，第三方应用开发商开发出了多种多样的应用场景。下面通过一些案例来帮助大家了解云存储Open API带来了什么样的机会。

有一类第三方应用通过连接多家云存储服务提供商的Open API来实现云存储集成服务，使得用户可以在一个应用中访问和管理多个云存储服务中的数据。这类应用的例子是ES文件浏览器。

另一类第三方应用专注于处理特殊格式的数据，并通过云存储服务提供商的Open API来保存这些数据。比如，Lucidchart专注于处理流程图，Smartsheet专注于处理项目管理，而PicMonkey专注于处理相片。它们都通过Google Drive的Open API来实现云存储的功能。

还有一类第三方应用专注于对电子内容进行二次处理，如对电子内容进行数字签名、传真电子内容、对电子内容进行分享等。此外，智能设备厂商（扫描仪、打印机、电子笔、智能电视）可以使用云存储Open API来为自己的智能设备添加云存储和云同步的功能。此外，浏览器（如海豚浏览器）和阅读器（如网易云阅读和网易新闻客户端）也通过云存储Open API为用户提供了知识碎片整理和收藏的功能。

我们相信在以开放与合作为主题的当下，云存储Open API是顺应潮流的。在三方共赢的前提下，更多有意义的用户场景将会基于Open API被实现出来。让我们拭目以待。

李崇欣

有道云笔记后台系统及OpenAPI的开发者，热衷于分布式系统性能、可靠性及扩展性的研究与开发，对HBase等NoSQL数据库有比较深入的了解。

蒋炜航

网易有道云笔记团队负责人。UIUC计算机博士。曾就职于HP Lab和NetApp，在硅谷参与创办Pattern Insight（已被VMware收购）。

责任编辑：杨爽（yangshuang@csdn.net）

从引擎到应用的云存储实战

百度云存储技术剖析

文 / 郭杏荣

本文从百度云存储的底层引擎Mola系统开始讲起，描述了如何在其上逐渐构建、扩展和完善，形成BCS、PCS和百度网盘等产品的过程，同时指出了当下云存储所面临的巨大挑战及应对之道。

个人云存储的时代浪潮

2011年，乔布斯在苹果开发者大会上发布iCloud时说：“过去数字生活的中心是PC，而未来数字生活的中心则是云。”是的，我们也看到人们越来越多地使用智能手机、平板电脑、智能电视和智能汽车电子设备，随时随地产生数据和访问数据。自然而然的，数据和围绕数据的服务，就要逐渐地转移到云端：数据在云中，服务也在云中，只有这样才能满足人们在众多智能设备间无缝衔接的数字化生活需求。

云存储的引擎——Mola系统

要构建云存储平台，首先需要一个可靠地高性能的存储引擎。Mola是我们研发的一个通用存储引擎，它是为高并发、低延时的在线服务而设计的，并且考虑到在线服务要求99.99%以上的稳定性，以及数据快速增长下的良好扩展性。按照分布式系统设计的CAP理论，Mola主要考虑数据的高可用性和分布性，数据一致性采用相对实用的最终一致性模型。图1给出了Mola的架构设计图。meta server存储的是数据的组织和分布信息。上层应用（App）调用molaSDK提交数据，数据经由update server发送到chunk server持久化存储。为了保证高可用性，数据采用3份冗余存储，读取数据时会按负载均衡选择一个chunk server上的副本。因此只要有一台chunk server仍存活，服务就可用。集群规模扩展可以通过增加新的chunk server来达到。Mola的设计体现了“简单可依赖”理念，但实际支持了百度搜索、百度音乐、百度地图等很多核心业务。

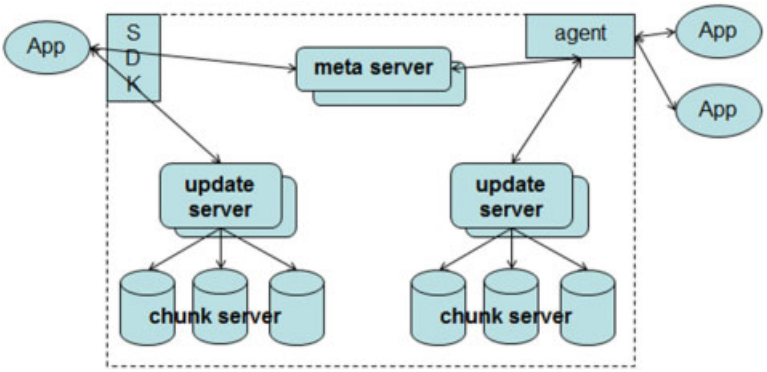


图1 Mola的技术架构

云存储新引擎

过去，为了存储不同类型和业务的数据，我们研发了诸多存储引擎。目前，我们正在把不同的存储引擎统一起来，这就是研发中的新存储。新存储的结构分为以下三层。

- 底层是存储介质层。除了硬盘，数据最终的存储媒介还有Flash/SSD存储。我们几年前就已逐渐试验这种设备，以获得更高的读写带宽和速度。另

外，对一些实时性要求非常高，但数据量不太大的计算，可以直接把数据和计算放到内存里面，一台机器不够，就用很多机器做分布式的内存数据结构。

- 中间是数据存储模型层，统一用块（block）来存储数据，即将文件、表格和对象都分解为block来存储。这样可以用一种基本的模型来支持不同的对象。
- 最上层是接口抽象层，对开发者提供一个一致的数据查询接口。

除了引擎本身的技术改造，云存储还要提供最快的速度，同时具有最高的经济性。为此，我们在华北、华东、华南建设核心存储集群，例如在山西阳泉建设的云计算数据中心设计存储规模达4000PB，相当于20多万个中国国家图书馆的藏书总量。围绕核心集群，在主要城市建设数量更多的CDN节点，把数据放到离用户的接入网络最近的地方（如图2所示）。

图2　新存储和存储集群部署

新存储系统将部署在核心存储集群和CDN节点上。CDN节点是内置于新存储系统的，这样可以做到当一个用户把某个文件分享出去形成访问时，这个文件会被自动发送到CDN节点。

云存储的发展：从BCS到PCS

前面介绍了存储引擎的发展，接下来的问题是：从一个存储引擎到开放的云存储，还需要考虑什么样的问题呢？对此我认为，云存储其实解决了通用存储引擎平台化和服务化的问题，也就是说用开放的服务来提供存储能力，支持多个用户（十万以上），规模要能够灵活地扩展到很大（百PB以上）。

在Mola的基础上，我们研发了面向开发者的开放云存储服务（Baidu Cloud Storage，简称BCS）。BCS为每个开发者提供一个独立的存储空间，开发者使用API来存取数据和控制数据的访问权限。建立在BCS之上的应用，像我们的自有产品一样稳定可靠。业界可与BCS类比的云存储服务还有Amazon S3等。

BCS是百度云存储发展的第一个阶段，在BCS的基础之上，我们进一步将云存储与移动结合起来，将云存储与个人直接结合起来，直接面向个人用户而不是企业开发者提供云存储，所以在2012年3月，我们推出了个人云存储（Personal Cloud Storage，简称PCS）。PCS是百度云战略的核心，正如本文开篇所述，PCS就是要给每个用户提供一朵云，这朵云里是用户专属的存储空间。用户在不同终端设备上所产生的数据，都可以保存在这个空间中；开发者基于PCS开发的各种应用，例如浏览器、输入法、音乐播放、文档阅读等，都可以把数据自动备份到PCS，与PCS同步。这样便可以实现用户在任何时间、任何地点、使用任何设备都可以访问自己的最新数据，使用围绕这些数据的服务。

有人可能会问，BCS和PCS有什么区别？这不仅是一个纯技术的问题，还是一个与产品和市场有关的问题。

第一个区别要从数据的存储方式来讲，PCS是我们把存储空间直接给到最终用户，每个用户可拥有15GB或更多的空间，而BCS则是开发者向我们租用或购买空间，再把空间分给各自的用户。

第二个区别是存储模型，BCS只能按Key-Value模型来存取数据，而PCS则实现了一个用户文件系统，每个用户看到自己的存储空间都是可以访问的文件和目录。

第三个区别是数据共享，这是非常关键的地方。在BCS上，数据是隔离的，A开发者的数据不能

被B开发者所访问，即便是同一个自然人的数据；PCS的数据是打通的，可共享的，因为开发者都是将数据存放在一个用户的空间下。这也是为什么PCS能解决数据变化和终端多样性带来的问题。

第四个区别要看用户成本，做一个数据类型的应用，存储和带宽的成本很高，对中小企业的开发者来说是很重的负担。就PCS而言，因为空间是我们分配给用户的，所以开发者不需要承担这个成本，一年可以省几百万元甚至上千万元。

最后的区别是对移动应用的优化，PCS通过文件格式转换、缩略图和增量更新，能够为移动设备节约流量和电量。图3给出了BCS和PCS的区别。

区别	BCS/S3/...	PCS
数据组织方式	按开发者	按个人用户
存储模型	Key-Value like	File-system like
数据共享	应用间隔离的	应用间打通的、可共享的
用户	不同的独立体系	共享百度5亿用户*
存储和带宽成本	开发者承担	百度和(或)用户承担
移动应用优化	无	有(如Delta; Preview)
	裸的 简单存储服务	完整的用户数据 云存储服务

*支持手机号、邮箱等方式登录

图3 BCS和PCS的区别

PCS的技术内幕

从技术实现上来说，PCS是在BCS的基础上发展起来的（如图4所示）。我们把PCS当做BCS上的一个大型开发者，给予无限制分配独立存储空间的权限。这样，PCS就能够为多达5亿的用户创建用户的专属存储空间。第二个工作其实是实现一个分布式文件系统（Bigtree）。假设有1亿用户，每个用户存一百个文件，这个文件系统的规模很容易达到百亿个文件之多。Bigtree把每个用户划分到一台服务器上，当应用请求一个用户的文件时，首先根据划分规则找到对应的文件系统服务器，依靠单机检索引擎即可快速查询到文件的属性。如果要读取文件内容，则根据文件属性中记录的Key到BCS去读取文件内容。

Quota模块则负责实时计算每个用户的实际使用空间，如果空间超过限制，则PCS会暂时禁止新的数据写入。Preview模块提供了计算图片不同尺寸的缩略图的服务。Metadata模块则用来存储文件

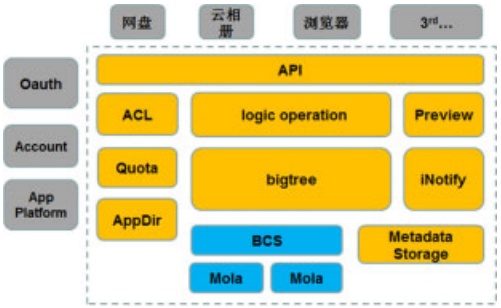


图4 基于BCS构建PCS

的标签（tag）信息，并提供标签检索。

PCS是一个平台，开发者可以开发任何需要保存和同步个人数据的应用。开发者将应用提交给我们，我们进行审核，通过后就可以在百度搜索、百度应用中心和PCS应用大全里出现，被百度的5亿用户所使用。第三方应用可以获得一个专有的目录（/apps/$APP_DIR）来保存应用的个人数据。例如一个叫camera（相机）的应用，可以把用户通过它拍的照片保存到/apps/camera下。如果第三方应用想要获得用户的其他文件，或者是其他应用所产生的文件，可以向用户申请图片、视频、文档和音乐类型的文件访问权限。用户通过OAuth协议来授权或取消授权。这体现了用户对自己的数据具有完全的控制力。如果有恶意的应用想要盗窃、滥用数据，用户可以禁止其访问云空间。

iNotify模块允许基于PCS的应用监听用户文件的改变，一旦有关心的变化，PCS就会通知这个应用，以便保持数据的同步。例如，当用户通过网盘上传了一张照片，云相册获得这个通知后就可以将这张照片加到用户最新的相册中。这也是数据打通共享的一个典型例子，就像今天在PC上，我们可以用不同的软件来打开和编辑照片和文档，而不必在意它们是由什么应用产生的一样。一个人的数据不应该被不同设备或者应用所分割。

实战：网盘如何基于PCS快速构建

有了PCS，一切变得很简单（如图5所示）。百度网盘的工程师只需要开发各个终端上的App。这些App都使用PCS的API/SDK，直接将文件和目录保存到PCS的用户空间下，目录结构保持一致。在手

图5　百度网盘基于PCS快速构建

机上，为了节约流量，网盘要靠用户的确认才会上传/下载文件；在PC上，网盘的常驻客户端进程可以监测用户的专用同步目录，当新建、修改和删除文件时，就会触发自动同步，将本地的修改及时保存到PCS；进程也会定期地检查PCS上是否有新的文件，如果有也会下载到本地，以保持数据的多端同步。为了把用户体验做到更好，网盘使用了一系列的算法和技术来加快数据的同步：

首先，使用增量通知机制来获得更新的文件，而不是全部文件列表的对比；

其次，当用户修改文件时，使用改进的rsync算法（由Andrew Tridgell提出的）来分析出实际改变的部分，这样就可以避免上传整个文件；

再次，利用我们已有的海量数据资源和其他用户已经上传的资源，进行基于指纹的比对（Cloud match），如果一个文件已经在PCS上，则用户就不用再真正上传，我们称之为“秒传”。

基于我们的技术优势，百度网盘还提供了离线下载功能，用户可能在网上有很多的资源想下载，但又没有时间，离线下载能够利用我们的服务器去在后台帮用户把资源下载并保存到PCS。通过我们的大规模数据计算和分析技术，系统知道哪些数据是最受欢迎的，就能够把相关的数据、音乐、文档和网页新闻等推荐给用户。在今天很多网盘产品一味地比拼空间的背景下，为用户提供一些不一样的有用的功能，无疑会更具特色。

云存储的最大挑战

当然，任何事物的发展都会遇到问题。移动云存储到今天也面临着一个首要困境：成本。

个人云存储的成本是多少？我们估计如果在两年以后PCS的用户数过亿，那么需要投入十几二十亿，而且以后还需要更多持续投入。在竞争的市场中，向用户直接收费还是一个充满挑战的课题。而且用户还可以选择其他的存储方案，例如U盘和移动硬盘的单位GB存储成本只有几分钱或者几毛钱。而我们看到今天每一个云存储服务商的价格都是很高的，iCloud、Dropbox达到每GB年12元左右，谷歌每GB年的价格接近4元。我们相信伴随着摩尔定律和技术的提升，明年或后年有可能降到2元、3元，但仍然是高成本。云存储的困境就在这里。如果没有非常强大的资本实力和变现能力，空喊口号提供几十、几百GB免费空间，都是噱头式的营销，难以长期持续，到最后人去楼空受伤的还是用户。

因此在面向个人的云存储市场，先发的优势并不存在，最重要的还是比拼长期的成本和价格。谁能够通过基础技术的架构优势把成本降下来，谁能够率先赢得大多数用户，并且能够有好的流量变现手段，谁才能笑到最后。

愿景：Powered by PCS

在PC时代，很多PC都贴着一个“Intel inside”的标识，告诉用户这台PC用的是Intel的CPU，值得信赖。这样的合作，对PC厂商和Intel来说是双赢。我们通过搭PCS这个“台”就是要帮助开发者更好地“唱戏”，我们希望未来有更多的应用“Powered by PCS”。

郭杏荣

就职于百度公司，先后从事搜索用户产品、互联网基础架构平台的研发，目前任移动·云事业部架构师，负责云平台的技术和产品研发。

责任编辑：杨爽（yangshuang@csdn.net）

云端的数据库

文 / 郭理靖

本文阐释了云端数据库的重要性，同时指出了数据库云服务的特性、技术原理以及实现。

现在，越来越多的企业开始使用云平台来搭建产品，而一般的产品都离不开数据库，且数据库几乎是所有产品的核心部件，因此各个云平台都推出了自己的数据库云服务，以方便用户更加快捷地部署自己的服务。例如，国外Amazon的RDS（Relational Database Service）、微软的SQL Azure、Google的CloudSQL、国内盛大云的数据库云服务、阿里云的RDS，以及新浪SAE的MySQL云服务。那么数据库云服务到底应该提供什么样的服务？能解决什么问题？其背后的技术思想又是什么呢？

服务特性

数据库云服务至少具有以下三方面的特性。

操作易用性

- 一键申请：用户应该可以很方便快捷地申请使用数据库云服务，只需在管理界面填入相关的参数，就可以很快地申请到一个可以使用的数据库，而不像传统做法，还需要用户自己安装配置数据库。数据库云服务甚至可以支持用户通过API的方式来申请数据库，方便用户的定制开发。
- 灵活申请：用户可以根据实际业务需求很灵活地申请不同类型的数据库，而不是只提供一种固定空间大小的数据库服务。当用户的业务增长时，可以提供快速升级数据库类型的功能。
- 统一管理：应该为用户提供一个完整的管理界面，可以让用户管理所有申请的数据库的生命周期，且能在管理界面上配置修改数据库参数，监控数据库的服务状态。

数据安全性

- 访问控制：提供类似IPTables/安全组的访问控制功能，可以方便地让用户指定哪些IP段可以访问数据库，同时在申请数据库时可以让用户指定使用哪个端口。
- 数据镜像：为用户提供随时可以对数据库做镜像（数据库备份）的功能，做完的数据镜像保存到云存储，保证镜像数据不会丢失，同时支持用户可以指定这个镜像生成新的数据库。
- 数据恢复：不管用户有没有手动备份数据库，云数据库服务都可以保证用户将自己的数据库恢复到七天内（或者更长/更短的时间）内的任意时刻，确保用户没有后顾之忧。

服务高可用

- 性能稳定：数据库云服务提供的数据库性能要比较稳定，不应该有太大的起伏，要有一致的、良

好的性能体验。同时要保证服务的可用性。

- 故障迁移：所有机器都会出问题，数据库云服务应该将故障自动迁移，而且故障迁移的过程对用户是不可见的，最好在故障迁移的过程中数据库服务一直可用，或者整个故障迁移过程非常快，基本上不影响用户的使用。

- 监控报表：用户可以查看完整的数据库监控图表，监控要非常完整地体现数据库的运行状态，同时用户可以查看数据库的使用报表，预测数据的增长量。

提供数据库云服务的核心思想应该是解决用户维护数据库的烦恼，就像雇了一个DBA来一样，帮你做数据库方方面面的事情。

技术原理与实现

现在我们来看看数据库云服务背后的技术原理与实现。在讲具体实现之前，先了解一下现在主流数据库云服务都有哪几种类型。

共享流

所谓的共享流是指在一台机器上启动一个数据库的Instance给多个用户使用，或者一台机器上启动多个Instance给多个用户使用，即多个用户共享一台机器上的资源。比如Google CloudSQL、SAE的MySQL和Amazon的DynamoDB服务等。

共享流有两个比较大的问题需要解决。

- 公平性：既然多个用户共享一台机器的所有资源，那么肯定会涉及到资源如何公平有效分配的问题。对数据库而言，一般都会使用磁盘、内存、CPU和网络这四个主要方面的资源。如何保证每个用户能公平合理地使用这些资源，对云服务而言是个不小的挑战。同时，以何种依据给用户分配不同比例的资源也是需要数据库云服务提供商仔细考虑的。数据库云服务提供商一般根据用户申请的数据库空间的大小来分配各种资源，将一台机器可用的硬盘空间与其他资源进行绑定。这种做法比较容易让用户理解，同时在实现方面有一定的便利性，但丧失了一定的灵活性，因为数据库空间的类型会相对变成固定几种。

同时，在公平性上也有两种做法：如果用户购买了某台机器上的10%的空间，那么此用户一直可以使用而且只能使用这台机器上的10%的资源；如果用户购买了某台机器上的10%的空间，那么此用户在最坏的情况下只能使用这台机器上的10%的资源，但当其他用户不太繁忙时，可以使用更多的资源。前者的实现会比后者更容易，但后者的利用率更高，而且可以给用户带来更多的实惠。值得一提的是Amazon的DynamoDB是采用带宽的方式来进行资源的绑定，这是非常巧妙独创的思路，值得其他厂商借鉴。

- 数据管理：如何给各个用户单独做数据库镜像？如何给各个用户提供数据恢复功能？当用户需要购买更大的数据库空间时，如何方便地让用户平滑快速地升级？这三个问题是所有数据库云服务提供商都应该考虑的。可能这些问题在独享流里实现方案比较简洁，但在共享型数据库流派里却是一个不小的难题。如果说，是在一台机器上启动多个Instance给多个用户使用，那么从理论上是可以采用独享型的技术来解决上述问题，只要保证不同用户使用不同的数据库端口就可以。

但在实际情况中，很少有数据库云服务提供商会采用这种方案。原因有两个：不管是哪种数据库，在正常启动时都需要额外的空间来保存信息，比如数据库的事务日志或者一些正常日志信息，而且需要的空间不少，那么同一台机器上可启动的Instance的数量（可以提供服务用户数）就没有单个Instance多；如果启动的数据库进程过多，各个进程切换的代价会比较大，而且需要对每个数据库进行监控，占用和浪费的资源也不少。因此从总的经济效益上来说，用单Instance模式会更加划算，但从开发的角度出发，采用多Instance的模式会更容易，只是后续会存在一些隐患。

独享流

顾名思义，独享流就是指只在一台机器上启动一个数据库的Instance给一个用户使用，即一个用户独占一台机器的所有资源。盛大云的数据库云和Amazon的RDS都允许用户按需求选择数据库的种类以及数据库空间的大小。实际上，独享流使用的机器可能不是真正的物理机，应该是虚拟主

机。因为如果使用物理机的话，就无法提供前面所提到的一键申请、灵活申请等服务。物理机的类型基本上是固定的，无法提供更小资源的数据库类型。而共享型的数据库服务不仅可以使用物理机也可以使用虚拟主机。

其实从某个角度上讲，如果使用虚拟主机，也是多个用户共享一台真正的物理机，但虚拟化技术会把虚拟主机对物理的资源共享性解决得非常好。因此可以认为一台虚拟主机就是一台独立的物理机。

一般虚拟主机都有多种类型可供选择，用户可以根据自己业务的特点需求来选择有足够CPU和内存的虚拟主机，但虚拟主机的本地硬盘是固定不变的。如果数据库单纯依赖于虚拟主机的话，那么也会丧失满足用户对不同数据库空间需求的灵活性。从服务高可用的角度来看，如果只使用一台虚拟主机来提供数据库服务，那么会存在服务单点，一旦这台主机出问题了，整个数据库就不可能用了。但如果使用两台虚拟主机采用Heartbeat+DRBD模式来提供数据库高可用服务的话，成本会成倍上升，对用户而言就显得太过昂贵了。有没有一种更加廉价的解决方案呢?

答案是有的，可以采用虚拟主机（EC2）+云硬盘（EBS）的模式，这样只需要使用一台虚拟主机，同时又有提供不同数据库空间（1GB到1TB，取决于云硬盘可以提供存储空间的大小）的灵活性，但会牺牲一点可用性。因为当虚拟主机出现问题时，重新申请一台虚拟主机，将云硬盘挂载上去需要一点时间，在这个期间整个数据库服务是不可使用的，但通常这个时间都会很短，一般用户都能够接受，因此牺牲短暂不可用的时间来节省一台虚拟主机的费用，是相对比较划算的。

不管是共享型数据库还是独享型数据库，都要对用户使用数据库的权限做一些限制，防止用户破坏云服务后台的数据管理。共享型和独享型数据库云服务的比较见表1。

从的技术角度来看，共享型的数据库云服务一般都需要去修改原来数据库的实现代码，因为原有的数据库服务并不是为了共享服务而设计的，不会考虑各个共享用户之间的资源公平性问题。但在资源公平性以及代码修改的实现难度的平衡上，各家又有不同的实现。

表1　共享型和独享型数据库云服务比较

	灵活性	性能	架构	价格	功能限制
共享型数据库	低	低	基于物理机	低	多
独享型数据库	高	高	基于虚拟机	高	少

实例

下面以盛大云的MongoIC为例，来探讨一下共享型数据库云服务的具体实现。

MongoIC是盛大云推出的一个MongoDB的云服务。目前有免费48MB的试用数据库和上限为100GB的收费数据库两种类型。

前面提到资源的公平性主要是从CPU、磁盘I/O、内存和网络这四方面来进行限制的。MongoDB采用file mapping来进行数据持久化，所有的数据文件都是映射到内存的，因此如果要对每个用户做单独的内存以及磁盘I/O限制的话，那需要对MongoDB做非常大的代码更改，从效益上来讲并不划算。确保每个用户公平地使用CPU也需要对MongoDB代码进行大改，很难实现。

那么MongoIC如何对资源进行公平性限制呢？答案是从以下三方面入手。

- 操作时间配额。MongoDB记录用户每个操作的时间和具体内容，这样可以方便用户找到操作中的慢查询，因此MongoIC会记录每个连接在5秒钟内所有操作累加的时间。当有新的请求时，MongoIC会查询一下是否已用完了此连接对应的操作时间配额。如果用完了，就会等待一段时间，等有新操作时间配额时再进行操作。从实现上讲，这和MongoDB本身结合比较紧密，而且需要修改的代码量不大，同时能保证资源的公平性。值得注意的是，MongoIC会分别对读、写操作做不同的时间配额。

- 网络带宽。MongoDB可以记录每个请求传入的数据量和传出的数据量，因此MongoIC可以很方便地统计每个用户的网络流量，对此做相应的限制。毕竟有时有些用户操作时间可能很短，但传输的数据量非常大，因而对磁盘I/O占用比较多，从而会影响到其他用户的使用。

- 连接数。每个用户连接都会占用系统的一定资

源，同时过多的连接会影响操作时间配额的配置，在用户大并发请求的情况下会造成大量闲置等待分配操作配额的连接，因此需要对不同的用户限制不同的连接数。

综上所述，操作时间配额对资源公平性的影响最大，是整个MongoIC资源公平性的基石，网络带宽其次，连接数的影响较少（如图1所示）。虽然主流Linux内核提供了控制组（Control Group）的功能，可以对进程做好各种资源的隔离，但MongoIC并没有选用这种方案，因为Control Group会对同个进程产生的所有线程做同样的限制，而我们只是想对用户连接的线程做资源限制，而不想同时影响MongoDB本身的一些工作线程。同时，还希望同台机器上的不同用户的连接操作配额分配是不相同的。之所以这样选择，主要还是实现难度、开发时间与资源公平程度的一个权衡。

本文前面也提到用户可用的资源很多时候与用户购买的数据库空间相关，MongoIC在前期并没有做非常严格的资源公平性，而且收费方式是根据用户使用硬盘空间以及使用索引来收费。这对MongoIC的收费模式与资源分配关联方面带来了很大的挑战。虽然可以定期扫描用户的数据库大小并修改相应的资源限制，但随着同一台机器上的所有用户数据的增长，就会面临本机硬盘空间不够的情况，此时有些用户数据需要平滑迁移到其他服务器上。这需要非常大的开发运维工作。

总结

独享型的数据库的实现基于对虚拟主机、云硬盘（EBS）以及云存储API的使用，同时对用户的权限做一些限制，一般不会对原有数据库代码进行修改。主要是利用虚拟化自带的资源公平技术来避免二次开发，同时保证动态申请、满足用户对不同计算资源的需求以及访问安全，利用云硬盘来保证存储资源的动态扩容、快速镜像以及故障转移，利用云存储来保存数据库的镜像和数据库操作日志而不用考虑容量与持久保证。

共享型数据库服务一般都是常年高可用，用户不用担心故障迁移等任何问题，但不是所有独有型数据库服务都是高可用的。比如当云主机出现故障时，Amazon RDS会出现短暂的停机，因为RDS需要申请新的云主机，挂载EBS，然后提供服务。当然这种可能性比较小，而且停机的时间也比较短暂，很多用户能接受。如果用户觉得难以忍受的话，可以申请Multi-Zone的服务（由两台云主机提供高可用服务），当然用户得付两倍的钱。因此用户也需要在可用性和价钱方面权衡一下。

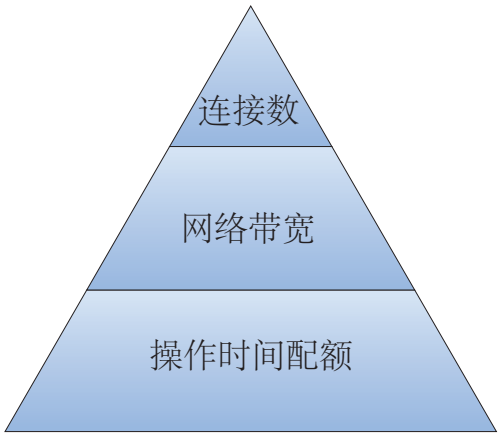


图1　各因素对资源公平性的影响

随着云计算的越来越成熟，云端的数据库服务也会变得多种多样，云平台不仅会提供现有数据库的服务（MySQL、MongoDB、SQL Server和Oracle等），也会提供云平台自有的数据库服务（DynamoDB、新浪的KVDB等），同时随着PaaS平台的普及，而共享型数据库服务是PaaS平台非常重要的一部分，所以共享型数据库服务也变得非常重要。未来云端的数据库云服务会共享型独享型花开两枝，齐头并进，与IaaS、PaaS平台形成良好的开发生态环境。

郭理靖

盛大云计算云数据库产品部经理，负责开发MongoIC、数据库云等云端的数据库服务。曾就职于金山、淘宝等公司。

责任编辑：杨爽（yangshuang@csdn.net）

高性能分布式复杂消息处理引擎

文 / 陆平，钱煜明，朱科支

本文阐述了一种高性能分布式复杂消息处理引擎的设计方案，该引擎克服了传统CEP处理引擎扩展性问题，同时讨论了在实施过程中的一些问题及解决方法。

物联网和互联网应用的共同特点是高并发、大数据量、动辄上亿的消息量。这不仅对消息处理的可靠性有一定的要求，对系统扩展性也有较高要求，因为在较短的时间内用户可能会从刚上线的几十万扩展到上千万甚至上亿。

海量实时数据的处理方法目前主要有两种：通过类似MapReduce的方法进行；将事件流化，直接在内存中进行海量数据的运算和处理。 第一种方法是micro-mapreduce，将MapReduce粒度变小，周期缩短，这种方法实时性稍差，但能够较好地解决可扩展性问题。第二种方法需要将相关数据载入内存并进行计算。现在有开源的流式处理框架如S4，商用的产品如Sybase CEP等。单机处理性能较高，但处理的可扩展性、容灾容错等会比较有问题。

Storm提供了比较好的分布式解决方案，Storm集群由一个主节点和多个工作节点构成，工作节点与主节点通过ZooKeeper协同工作。Storm本质上是一个可靠的分布式消息处理引擎，能保证每条消息都能够被处理。但其缺点在于主节点可靠性问题，以及对事件窗口及多路事件协同（例如发生事件A，如果同时过去30秒发生过事件B则生成新的事件C）没有比较好的支持。

为达到可靠处理海量实时数据的目的，我们开发了一套全新的高性能分布式复杂消息处理引擎ZX-CEP，重点实现了以下功能。

- 复杂事件数据的流式处理。
- 高并发，单机支持每秒十万以上的消息量，线性扩展能力较强。
- 实现简单的EPL（Event Process Language）消息处理编排以及图形化处理流程编排。
- 分布式计算，系统容量及处理能力能线性扩展。
- 滑动事件窗口。

分布式流计算架构

从系统层面，可认为分布式流计算系统是一个处理黑盒。大量连续的数据流进入黑盒，经过处理，转换为特定的事件流输出或传送到其他系统进一步处理。例如告警事件，可以生成动作事件到对应的执行机构进行操作，也可以将分析后的事件存储到持久化存储引擎以供后续分析处理。

流计算系统内的数据流向本质上是有向无环图。当 需要对数据进行多重处理时，我们可以将一个流程的输出作为另一个流程的输入，实现多个流程的序列化处理。

分布式复杂消息处理引擎的架构如图1所示。该系统由以下几个关键网元构成。

数据预处理模块

连续事件流传送过来的是各种未经过结构化处理的事件序列，通过事件预处理模块实现原始事件的过滤、合并和分流。

预处理模块分为两部分。

- 输入数据适配器，用于接收原始事件序列并转换为结构化事件，按事件发生的先后顺序送入本

地消息队列，等待数据预处理。一般要根据输入内容定制开发输入适配器。输入内容经过输入适配器后，被转换为标准的消息体格式，即消息源ID、消息发生时间戳和消息内容K/V对象。对于无法量化的消息，我们还需要有进行元数据管理，将消息内容进行量化处理映射。

- 预处理操作，即对事件预处理的一些原子操作，如字段过滤原子、字段填充原子、事件过滤原子、事件合并原子、事件拆分原子等。我们用任务管理器实现了基本原子操作的规则定制以及动态加载。基本原子操作可以实例化为多个算子，各个算子按照定义好的规则进行连接，就可以实现对数据的预处理。各个算子的连接方式，可以用图形化编辑工具生成，也可以通过类EPL语言的条件解析产生。对算子操作进行管线化连接的好处是，可以随时对基本算子进行各种串并联操作，实现复杂的数据处理逻辑而不需要编写复杂的代码。

在事件处理的过程中，输入信号有可能产生一些超出正常幅度之外的噪音信息，过滤操作能够有效去除噪音，保留正常信号。

复杂消息处理模块

多个事件处理模块侦听同一个或多个窗口的变更事件队列，空闲的事件处理模块自动从队列中获取待处理事件（如图2所示）。由于事件处理模块本身是无状态的，所以可以随时根据业务情况增加或减少事件处理模块而不会影响系统的运行。

本系统的事件处理模块基于规则引擎完成事件的基本处理。我们实现了一些对事件窗的基本的事件处理操作。

- Total：按条件计算事件窗内某变量的总和。
- Count/Condition Count：按照某条件计算事件窗内事件的数量。
- Average：按照条件求取事件窗内某个变量的平均值。
- Exist：按条件判断事件窗内是否存在某事件。
- Non Exist：按条件判断事件窗内是否不存在某事件。

对于传统的事件窗操作，对于这些基本算法均需要通过遍历事件窗内的数据来完成。但对于分布式系统，数据访问的开销比较大，每次运算均遍历数据基本不可行，因此要根据队列的输入/输出数据来进行计算。

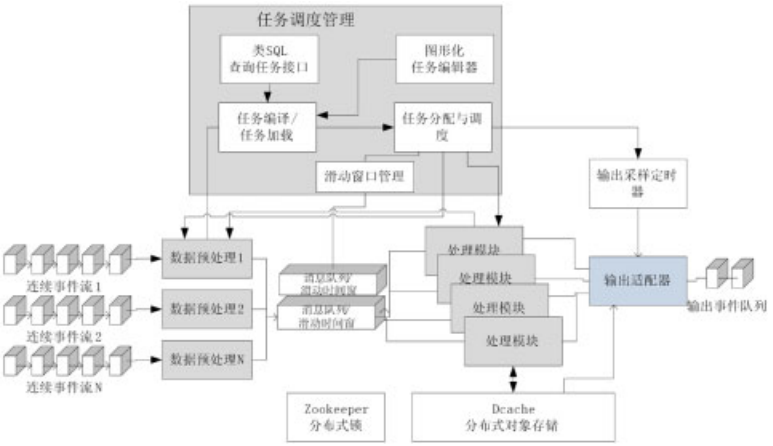


图1　分布式复杂消息处理引擎架构图

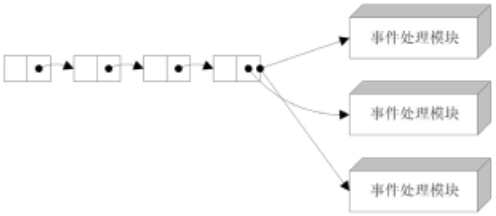


图2　多个事件处理模块分布式处理同一变更队列内容

输出适配模块

输出适配模块用于将系统处理结果转换为特定的输出动作或数据流。输出适配模块有两个基本类型：消息输出和定期采样输出。当以数据流方式输出时，输出的数据流可以作为输入流由另一组规则进行后续处理。

这种场景比较常见于需要根据不同维度进行分析，且要细粒度分析5分钟内数据流的情况。输出整合结果后形成粗粒度数据流，再进行更长时间段范围内的分析，如1天，可输出文件、数据库表或消息队列等多种形式。一般要根据业务需要定制开发输出适配模块。

输出适配模块还有一个功能是时光穿梭，即当规则条件被触发后，可通过输出适配模块记录在事件发生前后系统的各种相关消息状况，并做镜像持久化存储，以便日后分析问题。

任务调度管理模块

任务调度管理模块的主要工作是任务的生成和加载（如图3所示）。我们可通过两种方式生成任务：一种方式是用EPL的事件处理语句，动态定制生成任务图；另一种方式是通过规则编辑器，以图形界面方式生成事件处理逻辑。

图3　规则的生成与加载

在业务过程中，需要动态的规则加载，即规则加载过程不能够影响正常的处理过程。EPL适合比较简单的规则场景，规则编辑器则适合比较复杂的规则场景。为了提升效率，我们做了图形化的规则编辑器，将规则图生成后直接转换为对应的代码并实现了程序代码的动态加载。

当指定一个滑动窗口适配新的规则时，存在如何匹配发生在规则生效前旧数据的问题。为此，我们定义了两种实现策略：一种策略是在新规则部署后将清空对应窗口数据，但这可能会导致数据有一定时间中断；另一种策略是记录新规则生效后的时间信息，在此期间内新旧两套规则同时计算，直到新规则生效后的数据出栈后，才正式启用新规则的计算结果。否则，一直采用老规则计算结果。这可能造成的影响是仅当T= Tw+1（w为时间窗宽度）时，新的规则才能够生效。

滑动窗口设计及脏数据污点传播

滑动事件窗是通过高性能分布式消息队列实现的。滑动事件窗要求事件序列在系统中保留固定的时间长度或者固定数量的消息，且随着时间推移，保持在该事件序列内的所有消息都维持在特定时间/长度范围内。因此滑动事件窗分为两类：一类是滑动时间窗，所有事件都维持在特定的时间区间内；另一类是滑动空间窗，预先定义好窗体内事件的容量，超出容量的事件自动出栈。

为了保证系统的分布式处理，我们采用分布式K/V引擎来维护滑动事件窗口，这样事件的一致性存储就由分布式K/V引擎来保障。

事件序列的每个元素，以及入栈指针和出栈指针均作为K/V对保存在分布式K/V引擎中，这样便可实现分布式的滑动事件窗口存储。其中，入栈指针和出栈指针使用了特定的同步操作模式来进行存取，保证了分布式环境下的数据一致性（如图4所示）。

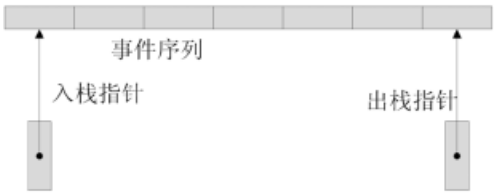


图4　分布式滑动事件窗口

每个事件进入事件窗或定时扫描发现是否有事件退出事件窗口，都会激发消息处理动作。该动作会激发复杂消息处理模块进行处理。为了保证处理的分布式，这里用消息队列方式实现了模式的消息处理。滑动事件窗的进入、退出事件会生成窗口变更消息，该消息会进入另一个消息队列等待复杂消息处理模块响应处理。

为此我们构建了类似Aurora[3]的数据模型，将时间窗的事件序列转换为增量事件序列。下面是三种增量事件。

- 插入事件：(+t)，t为新增到事件窗口的事件。
- 删除事件：(- , t)，t为从事件窗口退出的事件。
- 替换事件：(^t1, t2)，t1为被替换事件，t2为新的事件。

通过处理增量事件，系统能够有效避免经常性的全局扫描事件窗，从而大大加速处理的进程。在事件信息中，我们还增加了Qos标识，并发送到不同优先级的队列中。这样可以保证高优先级事件被优先处理。

我们利用分布式K/V存储维护事件状态机和全局计数器。在事件处理过程中，有效简化了数据的处理逻辑。

最简单的事件窗运算：计算事件窗内所有事件的平均值。普通方法是每次平均值都需要运算：

$$T(k) = \left(\sum_{i=0}^{n} E(k)\right)/n$$

而通过增量事件后，每次需要计算：

$$T(k+i)=T(k)+\sum_{m=k}^{k+i-1}E(m)/i$$

如果采样周期为1秒，事件窗为5分钟，那么后一种方法的计算量就只有前一方式的1/300。

分布式计算带来了一个问题：复杂的计算有可能涉及多个事件序列，多个事件队列产生的事件并不一定由同一个事件处理器处理。为此，我们引入了计算的污点数据传播模型，保证任何一个基础事件带来的信息更新都能够及时引发后续处理节点的处理。

当涉及到某个规则需要使用两个或多个滑动窗口内的数据时，分布式处理可能会导致两个滑动窗口产生的事件流并不是在同一个节点上进行分析处理的。为此我们设计了分布式的污点数据传播机制（如图5所示），保证每个规则数的各个处理节点都能够对最终结果进行正确更新，即使不是在同一个节点完成的计算。

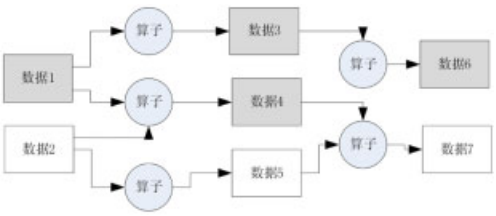


图5　污点传播的动态数据运算

整个规则树可以认为是一个有向无环图，变化后的数据，经过算子后，有可能影响后续处理节点，也有可能没有影响。动态分析能够查询出所有被影响的节点。静态分析仅能够分析出可信边界，可信边界外部数据可能是被污染的（Tainted），也可能是疑似污染的（Vulnerable），但可信边界内部的数据一定是干净的（Untainted）。

应用污点传播算法，可以识别出受影响的节点，并能通过算子重新计算受影响节点的后续数据。而对于没有受影响的数据则不需要重复计算。

在分布式系统中，每个数据以及算子，都有可能保存在不同的物理节点。

污点传播的分析有静态分析与动态分析两种，本系统实现了编译基本的静态分析。当源数据发生改变时，能分析出后续有向图中所有被影响的节点，并标示出被影响数据。当需要获取被影响数据进行计算时，要根据被污染标识，进行前向逆推计算。这保证了整个系统的计算量最小。

污点检查策略首要任务就是分析trust boundary（可信边界），它可以被表示为一个由实体类型type、脆弱性描述vul、程序操作op以及操作数位置loc组成的四元组：

type，vul，op，loc|type∈ROLES，vul∈VUL_TYPES，op∈ACTS，loc∈{N∪any}，

对每个输入变量对应的每个计算节点进行污染检查，就可以整理出系统的污染传播矩阵：

输入变量	被污染节点	疑似污染节点
A	T1，T2	V1
B	T2	V1，V2

对于多输入变量环境，被污染节点和疑似污染节点是单输入变量的并集。通过污点传播算法，可以让系统只在需要输出数据时对脏数据节点进行数据更新计算操作，而不需要时刻全面更新系统数据节点。这能够极大降低系统的计算量。

总结

本文描述了ZX-CEP分布式复杂消息处理引擎的设计及实现，该引擎能够高性能实时处理复杂的流式数据。我们首先基于数据与逻辑分离的原则对该系统进行了设计，数据存储节点采用云存储方式，保留多副本。数据处理节点采用无状态节点，可以分布式动态进行扩展。这种架构既保证了海量数据下的存储可扩展性和数据安全性，也保证了并行处理下的计算可扩展性。通过仿真测试表明在1000个节点的环境下，本系统最多能够进行每秒数千万次的事件处理。这个架构也保证了任意一个节点故障对于系统业务正常处理没有任何影响，流式计算仍然能够持续进行而不会中断。

未来我们将致力于进一步提升本系统的动态逻辑处理机制，让逻辑判断更加灵活，支持更加复杂的逻辑运算。同时我们将提升本系统的可维护性，使系统能够自动发现故障，并通过调整数据存储及计算节点实现故障的自我修复。

（本文三位作者均来自中兴通讯业务研究院。）

一种支持自由规划的Sharding扩容方案

主打无须数据迁移和修改路由代码

文 / 耿立超

在实现Sharding所需解决的一系列问题中，数据库扩容时一个热点话题。本文在深入探讨数据库扩容的基础上，提出了允许自由规划并能避免数据迁移和修改路由代码的Sharding扩容方案。

作为一种数据存储层面上的水平伸缩解决方案，数据库Sharding技术由来已久，很多海量数据系统在其发展演进的历程中都曾经历过分库分表的Sharding改造阶段。简单地说，Sharding就是将原来单一数据库按照一定的规则进行切分，把数据分散到多台物理机（我们称之为Shard)上存储，从而突破单机限制，使系统能以Scale-Out的方式应对不断上涨的海量数据，但这种切分对上层应用来说是透明的，多个物理上分布的数据库在逻辑上依然是一个库。实现Sharding需要解决一系列关键的技术问题，这些问题主要包括：切分策略、节点路由、全局主键生成、跨节点排序/分组/表关联、多数据源事务处理和数据库扩容等。关于这些问题可以参考我的博客专栏：http://blog.csdn.net/column/details/sharding.html。本文将重点围绕“数据库扩容”进行深入讨论，并提出一种允许自由规划并能避免数据迁移和修改路由代码的Sharding扩容方案。

Sharding扩容——系统维护不能承受之重

任何Sharding系统，在上线运行一段时间后，数据就会积累到当前节点规模所能承载的上限，此时就需要对数据库进行扩容了，也就是增加新的物理结点来分摊数据。如果系统使用的是基于ID进行散列的路由方式，那么团队需要根据新的节点规模重新计算所有数据应处的目标Shard，并将其迁移过去，这对团队来说无疑是一个巨大的维护负担；而如果系统是按增量区间进行路由（如每1千万条数据或是每一个月的数据存放在一个节点上），虽然可以避免数据的迁移，却有可能带来“热点”问题，也就是近期系统的读写都集中在最新创建的节点上（很多系统都有此类特点：新生数据的读写频率明显高于旧有数据），从而影响了系统性能。在这种两难的选择下，Sharding扩容显得异常困难。

一般来说，“理想”的扩容方案应该努力满足以下几个要求。

- 最好不迁移数据（无论如何，数据迁移都是一个让团队压力山大的问题）。
- 允许根据硬件资源自由规划扩容规模和节点存储负载。
- 能均匀地分布数据读写，避免“热点”问题。
- 保证对已达到存储上限的节点不再写入数据。

目前，能够避免数据迁移的优秀方案并不多，相对可行的有两种。一种是维护一张记录数据ID和目标Shard对应关系的映射表，写入时，数据都写入新扩容的Shard，同时将ID和目标节点写入映射表；读取时，先查映射表，找到目标Shard后再执行查询。该方案简单有效，但

读写数据都需要访问两次数据库，且映射表本身也极易成为性能瓶颈。为此系统不得不引入分布式缓存来缓存映射表数据，但这样也无法避免在写入时访问两次数据库，同时大量映射数据对缓存资源的消耗以及专门为此而引入分布式缓存的代价都是需要权衡的问题。另一种方案来自淘宝综合业务平台团队，它利用对2的倍数取余具有向前兼容的特性（如对4取余得1的数对2取余也是1）来分配数据，避免了行级别的数据迁移，但依然需要进行表级别的迁移，同时对扩容规模和分表数量都有限制。总地来说，这些方案都不是十分理想，多多少少都存在一些缺点，这也从一个侧面反映出了Sharding扩容的难度。

取长补短，兼容并包——一种理想的Sharding扩容方案

如前文所述，Sharding扩容与系统采用的路由规则密切相关：基于散列的路由能均匀地分布数据，却需要数据迁移，同时也无法避免对达到上限的节点不再写入新数据；基于增量区间的路由天然不存在数据迁移和向某一节点无上限写入数据的问题，却存在“热点”困扰。我们设计方案的初衷就是希望能结合两种路由规则的优势，摒弃各自的劣势，创造出一种接近“理想”状态的扩容方式，而这种方式简单概括起来就是：全局按增量区间分布数据，使用增量扩容，无数据迁移，局部使用散列方式分散数据读写，解决“热点”问题，同时对Sharding拓扑结构进行建模，使用一致的路由算法，扩容时只需追加节点数据，不再修改散列逻辑代码。

原理

首先，作为方案的基石，为了能使系统感知到Shard并基于Shard的分布进行路由计算，我们需要建立一个可以描述Sharding拓扑结构的编程模型。按照一般的切分原则，一个单一的数据库会首先进行垂直切分，垂直切分只是将关系密切的表划分在一起，我们把这样分出的一组表称为一个Partition。

接下来，如果Partition里的表数据量很大且增速迅猛，就再进行水平切分，水平切分会将一张表的数据按增量区间或散列方式分散到多个Shard上存储。在我们的方案里，使用增量区间与散列相结合的方式，全局上，数据按增量区间分布，但每个增量区间并不是按照某个Shard的存储规模划分的，而是根据一组Shard的存储总量来确定的，我们把这样的一组Shard称为一个ShardGroup；局部上，也就是一个ShardGroup内，记录会再按散列方式均匀分布到组内各Shard上。这样，一条数据的路由会先根据其ID所处的区间确定ShardGroup，然后再通过散列命中ShardGroup内的某个目标Shard。在每次扩容时，我们会引入一组新的Shard，组成一个新的ShardGroup，为其分配增量区间并标记为“可写入”，同时将原有ShardGroup标记为“不可写入”，于是新生数据就会写入新的ShardGroup，旧有数据不需要迁移。同时，在ShardGroup内部各Shard之间使用散列方式分布数据读写，进而又避免了“热点”问题。

最后，在Shard内部，当单表数据达到一定上限时，表的读写性能就开始大幅下滑，但整个数据库并没有达到存储和负载的上限，为了充分发挥服务器的性能，通常会新建多张一样的表，然后在新表上继续写入数据，我们把这样的表称为“分段表”（Fragment Table）。不过，引入分段表后所有的SQL在执行前都需要根据ID将其中的表名替换成真正的分段表名，这无疑增加了实现Sharding的难度，如果系统再使用了某种ORM框架，那么替换起来可能会更加困难。目前很多数据库提供一种与分段表类似的“分区”机制，但没有分段表的副作用，团队可以根据系统的实现情况在分段表和分区机制中灵活选择。

总之，基于上述切分原理，我们得到如图1所示的Sharding拓扑结构的领域模型。

在这个模型中，有几个细节需要注意：ShardGroup的writable属性用于标识该ShardGroup是否可以写入数据，一个系统在任何时候只能有一个ShardGroup是可写的，这个ShardGroup往往是最近一次扩容引入的；startId和endId属性用于标识该ShardGroup的ID增量区间；Shard的hashValue属性用于标识该Shard节点接受哪些散列值的数

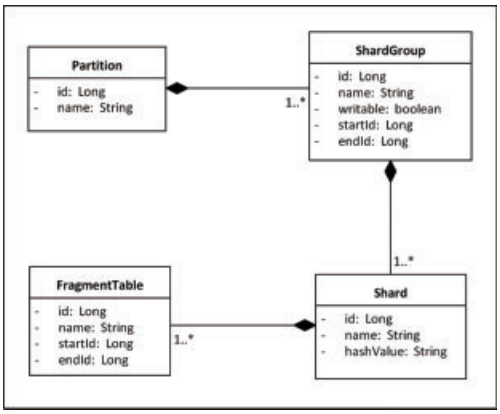


图1 Sharding拓扑结构领域模型

据；FragmentTable的startId和endId是用于标识该分段表储存数据的ID区间。

确立上述模型后，我们需要在数据库中建立与之对应的表来存储节点元数据，这样，整个存储系统的拓扑结构就可以持久化到数据库中，系统启动时就能从数据库中加载出当前存储系统的拓扑结构进行路由计算了，扩容时只需要向对应的表中加入相关的节点信息重启系统即可，不需要修改任何路由逻辑代码。

示例

让我们通过示例来了解这套方案是如何工作的。

■ 阶段一：初始上线

假设某系统初始上线，规划为某表提供4000万条记录的存储能力，若单表存储上限为1000万条，单库存储上限为2000万条，共需2个Shard，每个Shard包含两个分段表，ShardGroup增量区间为0-4000万，按2取余分散到2个Shard上，具体规划方案如图2所示。

与之相适应，Sharding拓扑结构的元数据如图3所示。

■ 阶段二：系统扩容

经过一段时间的运行，当原表总数据逼近4000万条上限时，系统就需要扩容了。为了演示方案的灵活性，我们假设现在有三台服务器Shard2、Shard3、Shard4，其性能和存储能力表现依次为Shard2<Shard3<Shard4，我们安排Shard2储存1000万条记录，Shard3储存2000万条记录，Shard4储存3000万条记录，这样，该表的总存储

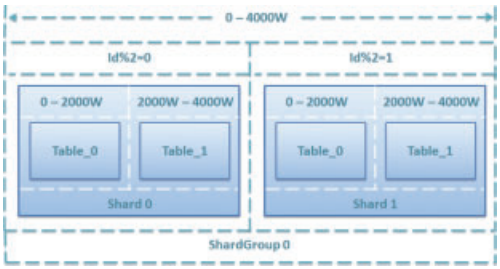


图2 初始4000W存储规模的规划方案

SHARD_GROUP

ID	NAME	WRITABLE	START_ID	END_ID
100	ShardGroup0	true	0	4000W

SHARD

ID	NAME	GROUP_ID	HASH_VALUE
1000	Shard0	100	0
1001	Shard1	100	1

FRAGMENT_TABLE

ID	NAME	SHARD_ID	START_ID	END_ID
10000	Table_0	1000	0	2000W
10001	Table_1	1000	2000W	4000W
10002	Table_0	1001	0	2000W
10003	Table_1	1001	2000W	4000W

图3 初始对应Sharding元数据

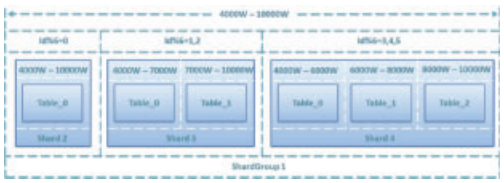


图4 二次扩容6000W存储规模的规划方案

SHARD_GROUP

ID	NAME	WRITABLE	START_ID	END_ID
100	ShardGroup0	false	0	4000W
101	ShardGroup1	true	4000W	10000W

SHARD

ID	NAME	GROUP_ID	HASH_VALUE
1002	Shard2	101	0
1003	Shard3	101	1,2
1004	Shard4	101	3,4,5

FRAGMENT_TABLE

ID	NAME	SHARD_ID	START_ID	END_ID
10004	Table_0	1002	4000W	10000W
10005	Table_0	1003	4000W	7000W
10006	Table_1	1003	7000W	10000W
10007	Table_0	1004	4000W	6000W
10008	Table_1	1004	6000W	8000W
10009	Table_2	1004	8000W	10000W

图5 二次扩容对应Sharding元数据

能力将由扩容前的4000万条提升到10000万条，图4是详细的规划方案。

相应拓扑结构表数据如图5所示。

从这个扩容案例中我们可以看出，该方案允许根

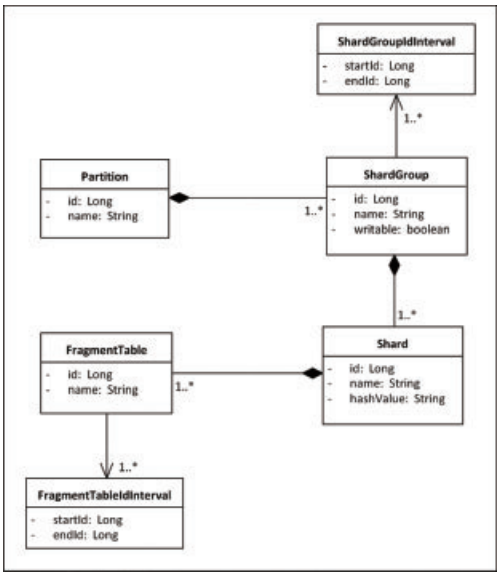


图6　增强后的Sharding拓扑结构领域模型

据硬件情况进行灵活规划，对扩容规模和节点数量没有硬性规定，是一种非常自由的扩容方案。

增强

接下来让我们讨论一个高级话题：对“再生”存储空间的利用。对于大多数系统来说，历史数据较为稳定，被更新或是删除的概率并不高，反映到数据库上就是历史Shard的数据量基本保持恒定。但也不排除某些系统其数据有同等的删除概率，甚至是越老的数据被删除的可能性越大，这样反映到数据库上就是历史Shard随着时间的推移，数据量会持续下降。在经历了一段时间后，节点就会腾出很大一部分存储空间，我们把这样的存储空间叫“再生”存储空间，如何有效利用再生存储空间是这些系统在设计扩容方案时需要特别考虑的。回到我们的方案，实际上我们只需要在现有基础上进行一个简单的升级就可以实现对再生存储空间的利用，升级的关键就是将过去ShardGroup和FragmentTable的单一的ID区间提升为多重ID区间。为此我们把ShardGroup和FragmentTable的ID区间属性抽离出来，分别用ShardGroupInterval和FragmentTableIdInterval表示，并和它们保持一对多关系。

让我们还是通过一个示例来了解升级后的方案是如何工作的。

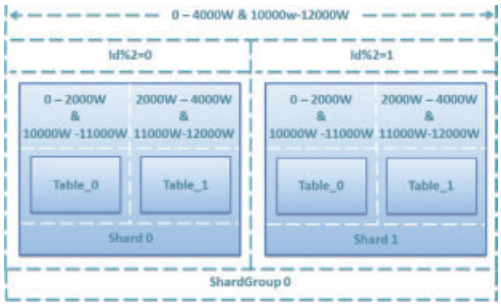


图7　重复利用2000W再生存储空间的规划方案

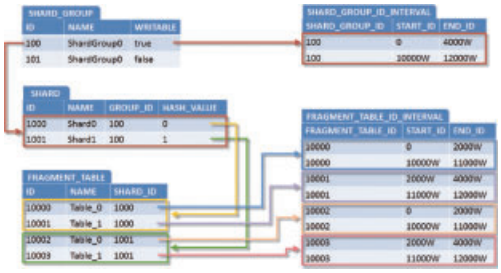


图8　重复利用后所对应Sharding元数据

■ 阶段三：不扩容，重复利用再生存储空间

假设系统又经过一段时间的运行之后，二次扩容的6000W条存储空间即将耗尽，但由于系统自身的特点，早期的很多数据被删除，Shard0和Shard1又各自腾出了一半的存储空间，于是ShardGroup0总计有2000W条的存储空间可以重复利用。为此，我们重新将ShardGroup0标记为writable=true，并给它追加一段ID区间：10000W-12000W，进而得到如图7所示规划方案。

相应拓扑结构的元数据如图8所示。

小结

这套方案综合利用了增量区间和散列两种路由方式的优势，避免了数据迁移和“热点”问题，同时，它对Sharding拓扑结构建模，使用了一致的路由算法，从而避免了扩容时修改路由代码，是一种理想的Sharding扩容方案。

耿立超

架构师，CSDN博客专家，目前主要从事企业级应用架构、SaaS、分布式系统和领域驱动设计相关的研究与开发工作，喜欢摄影和旅行。

责任编辑：卢鹄翔（ludx@csdn.net）

ThinkPHP 3.0架构设计解析

文 / 刘晨

ThinkPHP（http://thinkphp.cn）是国内知名的轻量级PHP开源框架，提供了完整的框架解决方案（例如ORM、Template、Cache、Session、Route、RBAC等），凭借简洁实用和快速开发的特性吸引了众多的PHP开发人员。本文剖析了ThinkPHP 3.0的架构设计思想，讲述其独到之处。

ThinkPHP的架构设计思想

ThinkPHP从诞生至今，参考了诸多国外优秀框架的设计思想，包括Struts、Rails、Django等，当然也不乏自己的一些改进和创新，而且敢于打破设计模式和开发准则。最新发布的3.0版本引入了全新的CBD开发模式，并支持SAE、REST和MongoDB，带给开发人员更好的使用体验。

图1 轻量级框架的设计原则

经过了6年多的完善和重构，ThinkPHP的编程思想和开发模式影响了很多PHP开发人员，框架的优劣不能仅凭功能的多寡来评判，架构和思想才是框架的灵魂，而ThinkPHP的优点体现在很多细节的设计之上，下面我将介绍该框架的一些主要设计思想。

MVC分层模式设计

ThinkPHP基于MVC（Model-View-Controller，模型-视图-控制器）模式，支持多层（Multi-Layer）设计。

模型（Model）层：默认的模型层由Model类构成，但随着项目的增大和业务体系的复杂化，单一的模型层很难解决问题，因而推出了对多层Model的支持，设计思路很简单，不同的模型层仍然都继承自系统的Model类，但在目录结构和命名规范上做了区分，例如某个项目设计中需要区分数据层、逻辑层、服务层等不同的模型层。我们可以在项目的Lib目录下面创建Model、Logic和Service目录，把对用户表的所有模型操作分成以下三层。

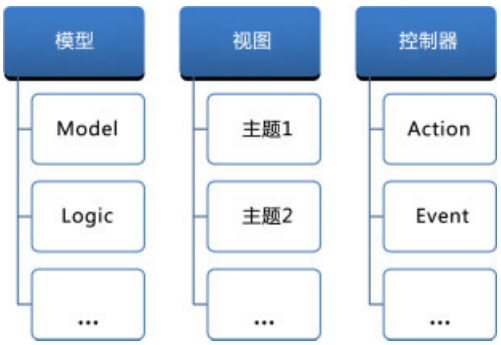


图2 Multi-Layer MVC设计

- Model/UserModel用于定义数据相关的自动验证、自动完成和数据存取接口。
- Logic/UserLogic用于定义用户相关的业务逻辑。
- Service/UserService用于定义用户相关的服务接口等。

而这三个模型操作统一继承Model类即可，这样对用户数据的操作就非常清晰，在调用时，也可以很方便地使用内置的M方法。

- D('User') 实例化UserModel。
- D('User','Logic') 实例化UserLogic。
- D('User','Service') 实例化UserService。

对模型层的划分不是强制的，开发人员可以根据项目的需要自由定义分层。

视图（View）层：由模板和模板引擎组成，在模板中可以直接使用PHP代码，模板引擎的设计会在后面讲述，通过驱动也可以支持其他第三方模板引擎。视图的多层可以简单通过目录区分，例如Tpl/default/User/add.html和Tpl/blue/User/add.html。

控制器（Controller）层：ThinkPHP的控制器层由核心控制器和业务控制器组成，核心控制器由系统内部的App类完成，负责应用（包括模块和操作）的调度控制，包括HTTP请求拦截和转发、加载配置等；业务控制器则由用户定义的Action类完成。新版增加了多层业务控制器的支持，其实现原理和模型的分层类似，例如业务控制器和事件控制器。

- Action/UserAction用于用户的业务逻辑控制和调度。
- Event/UserEvent用于用户的事件响应操作。

UserAction负责外部交互响应，通过URL请求响应，例如http://serverName/User/index，而UserEvent负责内部的事件响应，并且只能在内部调用A('User','Event')，因此是与外部隔离的。多层控制器的划分也不是强制的，可以根据项目的需要自由分层。控制器可以根据需要调用分层模型，也可以调用不同目录的视图模板。

在MVC三层中，ThinkPHP并不依赖M或者V，甚至可以只有C或只有V，这在ThinkPHP的设计中是一项很重要的用户体验设计，用户只需要定义视图，在没有C的情况下也能自动识别。

CBD和AOP编程思想

ThinkPHP是一个轻量级的开发框架，功能的完善和增加不会造成自身臃肿，因为它从3.0开始引入了CBD（核心+行为+驱动）模式，把框架的核心尽量做到最小化（这里的核心是指核心类库，目前ThinkPHP的核心仍然保持在100KB左右），很多功能特性都通过行为和驱动等扩展来完成。如果仔细分析新版的核心框架，会发现核心的Lib目录下多了Behavior和Driver目录，并且与Core目录处于同一级别，这就是新版内置的一些行为和驱动扩展。这些行为和驱动都是可替代的，如果开发人员没特别的要求，直接利用内置的核心扩展即可。

CBD模式的一个核心基础是系统的“切面”，行为都是围绕这个“切面”来进行编程的，这点非常类似于AOP编程思想。

图3 CBD模式构成

系统内置的切面包括：app_init（应用初始化）、path_info（PATH_INFO检测）、route_check（路由检测）、app_begin（应用开始）、action_name（操作方法）、action_begin（控制器开始）、view_begin（视图输出开始）、view_template（视图模板解析）、view_parse（视图解析）、view_filter（视图输出过滤）、view_end（视图输出结束）、action_end（控制器结束）、app_end（应用结束）等，开发人员还可以在应用中随意添加自己的切面。

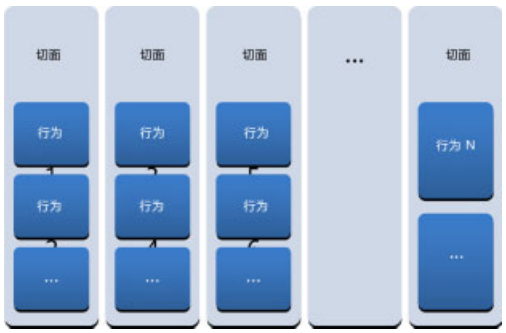


图4 CBD模式的“切面”思想

模式和引擎扩展设计

如果把系统内置的核心扩展看成是一种标准模式，那么用户可以把这一切的行为定制打包成一个新的模式，因此，在ThinkPHP中自然就有了模式扩展的概念。事实上，模式扩展不仅可以替换和增加行为，还可以对底层的MVC进行替换和修

改，以达到量身定制的目的。利用这一新特性，开发人员可以方便地通过模式扩展为自己量身定制一套特有的开发框架。

ThinkPHP 3.0还增加了引擎扩展方式，当模式扩展无法满足运行环境的特殊要求时，就需要使用引擎扩展。引擎扩展的一个最大特点就是替换了原有的框架入口文件，这是为了满足日益发展的App Engine平台而设计的。

其中一个典型就是SAE扩展，由于新浪SAE环境的特殊要求，内置标准模式中的文件写入操作无法在SAE环境中使用，因而导致系统框架入口ThinkPHP.php中的编译缓存无法实现，必须通过引擎扩展替换框架入口才可以实现。

文件导入及自动加载设计

ThinkPHP的文件导入传承了Java的类库包（Package）思想，开发过程中不再需要使用include或者require导入一长串的文件名，文件导入主要包括函数文件导入（load方法）和类库文件导入（import方法），并且都采用了命名空间的导入思想，如下所示。

```
// 导入ORG类库包的Util/Image.class.php类库
import('ORG.Util.Image');
// 导入项目的Common/Util/Array.php函数文件
load('@.Util.Array');
```

导入命名空间中的@表示当前项目，这个设计思想是为了便于类库和文件的跨项目移植。import方法内部调用了一个require_cache方法，确保即使存在多次导入也不会冲突。在impot方法之上，系统还为某些经常需要的导入操作进行了封装，例如实例化模型和控制器的操作肯定首先需要导入相关类库，因此A方法和D方法在内部会调用import方法，专为导入第三方类库而实现的vendor方法也是对import方法的调用。

开发人员基本无须手动导入类库，因为框架提供了可配置路径的自动加载功能。自动加载机制使用了spl_autoload_register（早期框架采用的是__autoload魔术方法，由于容易导致冲突已废弃），自动加载的实现方法在Think类的autoload方法中定义。

再来说说自动加载机制的设计思路。自动加载机制是真正的按需加载，可以大幅提高性能。自动加载有三种情况，按照加载优先级从高到低分别是：别名自动加载、系统规则自动加载和自定义路径自动加载。别名是预先给某个类库定义好的别名，它的优先级最高。也可以起到拦截系统自动加载规则的作用。系统对项目中最常用的Action类、Model类和行为类是可以自动加载的，因此这些类可以直接继承和实例化，而不用手动导入。如果你有额外的需要，却又不想预先定义别名，那么就可以修改自动加载路径配置APP_AUTOLOAD_PATH参数，这三种方式的结合，有效保证了自动加载的灵活性和性能的平衡。

函数重载设计

严格来说，PHP是不支持传统重载的，不过可以利用PHP弱数据类型、默认参数和可变传参的特性来实现“伪重载”功能（同名的函数，但函数的参数类型和个数不同、甚至不同的值完成不同的功能），这也是ThinkPHP“大道至简”理念的一个非常典型的体现，框架提供的全局函数中有很多类似的用法设计，包括C函数、cache函数和session函数等。开发人员不用记忆太多的方法名称。举个简单的例子，以新版的session函数为例，函数定义了两个参数session($name, $value=''),但可以支持下面5种（其实还有更多）用法：

```
// Session初始化
session(array('name'=>'session_id','expire'=>3600));
session('name','value');  // 设置session
$value = session('name'); // 读取session值
session('name',null); // 删除session值
session(null); // 清空当前的session
```

URL模式和路由设计

由于ThinkPHP框架的应用采用单一入口文件执行，所以网站的所有模块和操作都通过URL参数来访问和执行，并且默认采用了PATHINFO URL模式，以获得更好的SEO效果，并同时为不支持PATHINFO的环境添加了兼容模式和普通模式两种支持。

如果对URL要求较高，可以使用路由功能完成更优雅的URL地址解析，ThinkPHP支持规则路由和正则路由两种定义方式，规则路由定义更简单，可以满足大多数的情况要求，例如：

```
'news/:year/:month/:day' => array('News/archive',
                                  'status=1'),
'news/:id'               => 'News/read',
```

而正则路由更强大，可以满足对URL的复杂要求，例如：

```
'/^blog\/(\d+)$/'        => 'Blog/read?id=:1',
'/^blog\/(\d+)_(\d+)$/'  => 'blog.php?id=:1&page=:2',
```

惯例重于配置的全局配置设计

对于一个框架或一个应用来说，“配置”是一个不可或缺的部分，当然也是让框架变得更加耦合的原因之一。ThinkPHP参考Rails引入了惯例配置的概念，让开发人员尽可能减少自己配置参数，定义方式采用PHP返回数组，而且全局可动态设置（在参数未生效的情况下）。为配置的动态存取单独定义了一个C方法，用以简化配置操作。

```
C('配置名称','配置值'); // 设置配置参数的值
C('配置名称'); // 获取配置参数值
```

编译缓存设计

编译缓存是ThinkPHP比较独特的一个设计思想，虽说原理很简单，但效果比较显著。编译缓存的基础原理是第一次运行时把核心需要加载的文件去空白和注释后合并到一个~runtime.php文件中，再次运行时直接载入编译后的文件而无需载入众多核心文件。之所以称之为编译缓存，是因为存在一个预编译的过程，包括一些环境变量和常量的硬编码、目录检测，以及自动生成、去掉编译后运行不依赖的代码等，因为节省了I/O开销和部分代码运行开销，所以执行速度有了明显的提升。

项目编译机制对运行没有任何影响，预编译操作和目录检测机制只会执行一次，因此无论在预编译过程中做了多少复杂的操作，对后面的执行没有任何效率的缺失，即使删除也会自动生成。

3.0版本对编译缓存做了一些改进，可以直接替换框架入口甚至网站入口，从某种程度来说，编译后的框架甚至可以脱离框架核心独立运行；还支持在编译缓存中载入外部的常量定义文件，便于产品做用户定义。

ORM和查询语言设计

ThinkPHP的ORM（Object/Relation Mapping，对象关系映射）主要由Model类、DB类和数据库驱动组成，不需要依赖第三方的ORM类库。ORM设计应该属于ThinkPHP最神奇的部分（没有之一），包括CURD、ActiveRecord、查询语言、视图、关联和一些高级数据库特性。

不建议在框架中直接写SQL语句（虽然也支持），因为这样就不能享受连贯操作的方便和灵活性，也无法体现内置查询语言的魅力了，下面是一个典型的连贯操作用法：

```
$News->where('status=1')->
order('create_time DESC')->limit(10)->select();
```

最终解析成的SQL语句等同于：

```
SELECT * FROM news WHERE status=1
  ORDER BY create_time DESC LIMIT 10
```

配合3.1新版增加的命名范围功能，更是如虎添翼，例如我们定义了命名范围latest：

```
class NewsModel extends Model {
protected $_scope = array(
    // 命名范围latest
    'latest'=>array(
    'where'=>array('status'=>1),
    'order'=>'create_time DESC',
    'limit'=>10,
    ),
);
}
```

则可以更简单地进行查询操作：

```
$News->scope('latest')-> select();
```

ORM内置的查询语言则实现了跨数据库的查询条件的统一处理：

```
// 快捷查询
$map['name|title'] = array('like','%thinkphp%');
// IN查询
$map['id']  = array('in','1,3,8');
// 区间查询
$map['score'] = array('between','60,100') ;
// 请求查询
$map['_query'] = 'status=1&score=100&_logic=or';
// 字符串条件
$map['_string'] = 'status=1 AND score>10';
$News->where($map)->select();
```

查询语言中还有一项特别的动态查询方法：

```
// 根据邮件地址字段进行查询
$User->getByEmail('liu21st@gmail.com');
// 根据用户名字段进行查询
$User->getByUsername('thinkphp');
```

可以根据数据表的任意字段进行动态查询。

除此之外，查询语言还可以支持组合查询、复合查询、统计查询、定位查询、多表查询，以及子查询等功能。

ORM中的数据库中间层解决了不同数据库的差异问题，目前已支持MySQL、SQLServer、PgSQL、Sqlite、Oracle、Ibase、MongoDB等，也包括对PDO的支持。数据库中间层还内置了多数据库和分布式支持。

ActiveRecord模式

ActiveRecord（活动记录）是一种领域模型模式，由Rails最早提出，遵循标准的ORM模型：表映射到记录、记录映射到对象、字段映射到对象属性，能够很大程度快速实现模型的操作，而且简洁易懂。这种ActiveRecord适合非常简单的领域需求，尤其在领域模型和数据库模型十分相似的情况。

ThinkPHP很早就将ActiveRecord模式引入到ORM体系中，而且充分利用了PHP5的动态特性，简洁优雅地实现了PHP的ActiveRecord模式。因此你不需要在模型中定义具体的数据表字段属性（例如上面的UserModel的name和email属性），因为系统具有自动获取模型对应数据表的字段功能（这个功能在部署模式下面是有缓存的，只需要获取一次）。其次，ActiveRecord模式的对象属性操作在ThinkPHP中通过__set和__get两个魔术方法动态实现，因此也不需要定义存取方法。

因此，所有对数据对象的属性操作都转换成了对模型的data属性的操作，剩下的就交给CURD方法的处理data属性的细节了，这里暂且不表。

总结

ThinkPHP 3.0很多独特的设计思想都体现在细节之中，由于篇幅有限，无法一一剖析。其实这些设计思想的背后实现都很简单，但正是这些看似简单的设计，组成了一个有细节的轻量级框架，造就了ThinkPHP的不平凡，同时也满足了开发人员对开发效率、可靠性、可维护性和可扩展性等方面的要求。这些设计思想的精髓就是：大道至简，开发由我。

刘晨

ThinkPHP开源框架创始人和领导者，上海顶想信息科技有限公司CEO。拥有12年开发和团队管理经验。主要研究领域包括Web应用架构和开发、用户体验设计及网站运营。

责任编辑：卢鹓翔（ludx@csdn.net）

秘密共享协议及其应用

文 / 顾森

互联网给人们的日常生活带来了便捷，同时也引发了一些意想不到的安全隐患。很多生活中常见的安全需求，用物理手段很容易解决；而一旦到了纯信息交互的网络世界中，事情就变得棘手了许多。为了解决这些问题，人们想出了各种机智巧妙的协议。在最近的几期杂志中，让我们一起来探讨一些有意思的协议问题。

秘密共享问题

设想这样一个场景：总部打算把一份秘密文件发送给5名特工，但直接把文件原封不动地发给每个人，很难保障安全。万一有特工背叛或者被捕，把秘密泄露给了敌人怎么办？于是就有了电影和小说中经常出现的情节：把绝密文件拆成5份，5名特工各自只持有文件的1/5。不过，原来的问题并没有彻底解决，我们只能祈祷坏人窃取到的并不是最关键的文件片段。因此，更好的做法是对原文件进行加密，每名特工只持有密码的1/5，这5名特工需要同时在场才能获取文件全文。但这也有一个隐患：如果真有特工被抓了，当坏人们发现只拿到其中一份密码没有任何用处的同时，特工们也会因为少一份密码无法解开全文而烦恼。此时，你或许会想，是否有什么办法能够让特工们仍然可以恢复原文，即使一部分特工被抓住了？换句话说，有没有什么密文发布方式，使得只要5个人中半数以上在场就可以解开绝密文件？这样的话，坏人必须要能操纵半数以上的特工才可能对秘密文件造成实质性的影响。这种秘密共享方式被称为(3, 5)门限方案，意即5个人中至少3人在场才能解开密文。

在物理世界中，这并不是一个难题。比方说，总部可以把绝密文件锁进一个特殊的装置里，装置上有三个一模一样的锁孔，并配有5把完全相同且不可复制的钥匙。只有把其中任意3把钥匙同时插进钥匙孔并一起转动，才能打开整个装置。将5把钥匙分发给5名特工，目的就达到了。问题在于，这个方法不能用于网络世界中。因而，这不算密码学意义上的解决方法。要想在纯信息交换的层面上实现秘密共享，我们还需要想点别的招儿。

组合数学方案

利用组合数学，我们可以漂亮地实现(m, n)门限方案。把这份文件的密码拆成C(n, m-1)份（其中C(i, j)是组合数记号，表示从i个物体中选出j个物体的方案数），每个人持有C(n-1, m-1)份密码。不妨假设文件的密码是一个100位数，那么在(3, 5)门限方案中，我们需要把这个密码拆成C(5, 2)=10份，每份密码都是一个10位数。不妨把这10份密码分别用0到9编号，把每份密码都额外复制两份。5名特工各持有6份密码，密码的分配如表1所示。

你可以自己验证一下：任意3名特工碰头，都能凑齐这10份密码；但任意2名特工碰头，都无法凑齐所有的密码。

上述分配表的构造其实很简单：给每个可能的

表1 密码分配表

特工1	0	1	2	3	4	5				
特工2	0	1	2				6	7	8	
特工3	0			3	4		6	7		9
特工4		1		3		5	6		8	9
特工5			2		4	5		7	8	9

“三人组”分配一份密码。从5个特工中选出3个人共有10种方案，因此我们正好要10份密码。例如把密码0分给特工1、2、3，把密码1分给特工1、2、4，一直到把密码9分给特工3、4、5。这样的话，任意2个人在场都无法打开文件，因为他们始终缺少一份密码（这份密码分给了其余3个人）。而任意3个人在场都足以打开文件，因为每一份密码都只缺少2个人的量，不可能出现这3个人都没有的情况。这样，我们便利用组合数学巧妙地解决了这一问题。

线性代数方案

在正统的密码学中，我们有一些更精妙的方案。最直观的方法是，把文件密码编码为三维空间中的一个点，然后生成5个过该点的平面，每个特工持有其中一个平面方程。显然，2个特工在一起是无法获得原文件的，因为2个平面的公共点有无穷多个；但3个平面的交点是唯一的，因此任意3个人在一起都能解开原文件。

另一个有趣的办法利用了下面这个事实：知道k-1次多项式函数上的任意k个点就能恢复出整个多项式。因此，我们可以把文件密码编码为一个二次多项式p(x)，然后把p(1)、p(2)、p(3)、p(4)和p(5)的值告诉对应的特工。任意3个特工碰头之后，只需要解出这个多项式即可恢复出文件的密码来。

上述两种方案的本质都是相同的：把文件密码设为3个数x、y、z，然后编写5个与x、y、z有关的一次方程，并把这5个方程分别交给5名特工。假如文件的密码是116.35、39.975、67.167这3个数，只有同时拥有这3个数，才能解开原文件。那么，我们就用这3个数编写5个三元一次方程：

$3.4x + 5.6y - 2.81z = 430.711$

$x - 2.11y + 0.09z = 38.0478$

$7x + 9.9y - 0.1z = 1203.49$

$-0.3x + 2.24y + 5.6z = 430.774$

$3x + 4.5y + 6.67z = 976.941$

其中x=116.35、y=39.975、z=67.167是它们的公共解。但要想确定出这个公共解，只有1个方程或者2个方程是不够的。事实上，至少需要3个方程，才能保证三元一次方程组存在唯一解。因此，至少需要3个人在场，才能获得秘密文件的密码。

数论方案

《孙子算经》卷下第二十六问：“今有物，不知其数。三、三数之，剩二；五、五数之，剩三；七、七数之，剩二。问物几何？答曰：二十三。”有趣的是，不管物体总数除以3的余数、除以5的余数以及除以7的余数分别是多少，我们总能在0到3×5×7-1当中找出一个解来；换句话说，如果把一个数除以3、除以5和除以7的余数分别记作r_3、r_5、r_7，那么前3×5×7个非负整数与所有可能的(r_3, r_5, r_7)的组合之间存在一一对应的关系。后人将其扩展为“中国剩余定理”：给出m个两两互质的整数，它们的乘积为P；假设有一个大整数M，如果我们已知M分别除以这m个数所得的余数，那么在0到P-1的范围内可以唯一地确定这个M。求出这个M相当于解一个线性同余方程组，这可以利用扩展的辗转相除法来实现。在这个M的基础上加上或者减去P的整数倍，可以得到所有其他满足要求的解。

我们可以想办法构造这样一种情况，n个数之中任意m个的乘积都比M大，但任意m-1个数的乘积都比M小。这样，任意m个除数就能唯一地确定M，但m-1个数就不足以求出M来。Mignotte门限

方案就用到了这样一个思路。我们选取n个两两互质的数，使得最小的m个数的乘积比最大的m-1个数的乘积还大。例如，在(3, 5)门限方案中，我们可以取53、59、64、67、71这5个数，前面3个数乘起来得200128，而后面两个数相乘才得4757。我们把文件的密码设为一个4757和200128之间的整数，例如123456。分别算出123456除以上面那5个数的余数，得到19、28、0、42、58。然后，把(53, 19)、(59, 28)、(64, 0)、(67, 42)、(71, 58)分别告诉5名特工，也就是说特工1只知道密码除以53余19，特工2只知道密码除以59余28等。这样，根据中国剩余定理，任意3名特工碰头后就可以唯一地确定出123456，但根据2名特工手中的信息只能得到成百上千个不定解。例如，假设我们知道了x除以59余28，也知道了x除以67余42，那么我们只能确定在0和59×67-1之间有一个解913，而M则是一个形如59×67×k+913的数，其中k的数量级和当初选的那5个数一样大。

秘密共享协议的应用

我们找到了很多秘密共享的门限方案，但这个问题的应用场景太古怪，似乎没有什么实际的应用。其实不然。很多基本的安全需求都可以归结为秘密共享问题。例如，在数据储存和数据传输中，我们经常需要面对有部分字节丢失的情况，此时我们会想要设计一种数据编码方案，使得即使丢失了一部分字节的信息，凭借剩余的字节也能把整个数据恢复出来。

1960年，Irving S. Reed和Gustave Solomon提出了Reed-Solomon编码，其基本思想之前已经提到：有限域上的k-1次多项式可以唯一地由k个点确定出来。注意，为了便于编码，这里我们把数值空间限定在了有限域里。通常我们会取一个大小为256的有限域，而单个字节正好有256种取值，这样我们便能把每个字节的值视为有限域中的一个元素。假如某段数据有k个字节，那么首先我们把这k个字节看作该有限域上的一个k-1次多项式的k个系数，从而确定出k-1次多项式p(x)；然后计算出255个非0元素依次代入多项式p(x)所得到的函数值，作为最终的编码发送出去。由于我们总是在有限域内进行运算，所以多项式的值仍然在这个域内，仍然可以用一个字节来表达。因此，最终编码就是一段包含255个字节的数据。在实际应用中通常取k=223，再按此大小分割原数据，这样的话每223个字节的原始信息都可以被重新编码为255个字节，使得其中任意32个字节丢失后，我们仍然能够复原出全部223个字节的原始信息。

与其他纠错编码相比，Reed-Solomon编码有个颇有些意外的优势：它承受成片数据丢失的能力更强。这是因为，Reed-Solomon编码是以字节为单位的，对于它来说，丢失1个bit的数据和丢失连续8个bit的数据没什么区别。因而，Reed-Solomon编码广泛用于CD、DVD等存储媒介。数据备份小程序rsbep（缩写自Reed-Solomon and Burst Error Protection）则把最终得到的每255个字节为一组的数据$a_1, a_2, \cdots, a_{255}, b_1, b_2, \cdots, b_{255}, c_1, c_1, \cdots, c_{255}, \cdots \cdots$ 按照$a_1, b_1, c_1, \cdots, a_2, b_2, c_2, \cdots, \cdots \cdots, a_{255}, b_{255}, c_{255}, \cdots$的方式重新排列后存储起来，故意把成组的255个字节分散到各个地方储存，从而能够承受更严重的大块数据丢失。QR二维码也采用了Reed-Solomon编码技术，因而即使二维码缺失了一个角，或者在二维码中央覆盖一个装饰性的Logo，整个二维码的内容仍然能被100%地识别出来。

当然，反过来，如果数据缺失的位置更多是零散地分布在各处，这里丢1个bit，那里丢1个bit，此时再用Reed-Solomon编码就有些亏了，采用以bit为单位的纠错编码则会更好一些。

顾森

网名Matrix67，北京大学中文系应用语言学专业学生，数学爱好者。2005年开办数学博客http://www.matrix67.com，至今已积累上千篇文章，已有上万人订阅。

责任编辑：卢鹄翔（ludx@csdn.net）

通过Falcon谈安全类软件的开发和开源

文 / 李恒磊

Falcon是一款开源文件安全监控程序，作者在文中介绍了它的架构和原理，并谈到了自己对安全软件加入开源社区的感受。

讲述Falcon自己的故事

Falcon的产生背景

Falcon的前身Webradar，是采用PHP实现的文件监控程序，源码地址是http://www.oschina.net/p/webradar，是我的朋友董方当年在搜狐工作时开发的。它的工作原理是第一次运行获取目录下所有PHP文件的MD5值，通过cron定时运行，对比文件MD5的变化来找出修改或者新增的文件，进而检查是否包含恶意代码。Falcon曾部署在搜狗的多台服务器上，而考虑到实时性、效率问题及处理流程的一次性，我们决定基于inotify开发C语言版本的监控程序。

Falcon的框架设计和开发流程

我们的总体设计思想是将Falcon设计成一个文件监控框架，方便其他开发人员或使用者能随心所欲地配置，来监控自己感兴趣的文件和系统信息。

具体的实现都是直接借鉴的比较成熟的技术，软件开发尽量避免重复发明轮子，而且成熟的技术开发起来难度不是很大，也不容易出现难以解决的Bug。利用inotify模块的实时监控文件变化的功能，借助inotify-tool灵活、强大的API进行开发，针对新增、修改和删除等事件，对文件进行有针对性的检查，发现webshell和可疑文件并处理。

程序的大致流程如图1所示。

从流程图可以看出，这个系统的整体逻辑很简单，主要在于如何判断webshell文件和敏感文件的特征。提取特征的细节我在下一节会通过一个具体的webshell的例子来讲解。

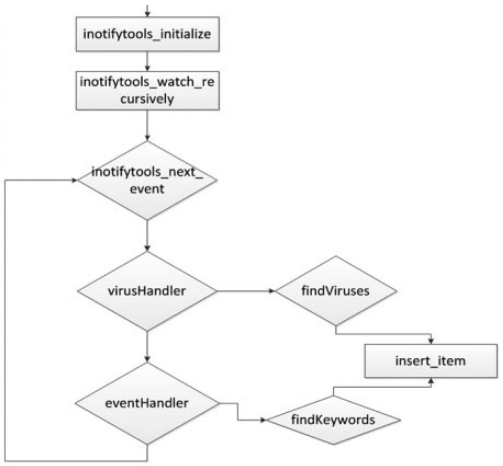


图1 Falcon程序流程

我们确定要开发这套系统时就确定了实时性是第一关注点，因此选定了inotify模块为我们的系统提供实时监控。在开发初期，我们接触了inotify-tool，它不仅提供了很好用的命令行工具，更提供了方便的API，对inotify进行了封装，使基于inotify的开发变得简单有趣。实时性的问题算是在技术层面基本解决了。

然后Falcon的功能性，因为这套系统是用来监控web目录下的文件安全性，所以就有了一个特定的功能点，针对这个功能点，我们主要关注新增和修改这两个事件。当这两个事件发生时，就会要调用webshell和敏感文件查杀的函数。先判断是不是webshell，是的话就入库相应的事件，若不是，则判断是否为敏感文件，处理流程和webshell基本一致，只有入库的事件有差别。由于Falcon是个监控系统，所以要在后台运行。

以上就是Falcon的整体开发流程了，下面将分析Falcon实现上的细节，便于大家进行二次开发，或者推倒重来。不管怎样，都希望能给大家带来一些启发。

Falcon的重要逻辑

我们将Falcon的源代码在GitHub上开源，网址为https://github.com/secrule/falcon，大家可以把代码下载到本地，对照着代码就不会被我说迷糊啦。好了，废话不多说，看着代码听我白话Falcon。

下面是第一段比较重要的代码，这段代码就是我们程序的主体逻辑所在。

```
struct inotify_event * event =
  inotifytools_next_event(-1);

while (event) {
  if (((strstr(event->name, ".php") != NULL
        && strstr(event->name, ".swp") == NULL
        && strstr(event->name, ".swx") == NULL
        && strstr(event->name, "~") == NULL)
       || (event->mask & IN_ISDIR))) {
    if (virusHandler(event, phpVirusfeature,
                     php_virus_num) != 1) {
      eventHandler(event, phpKeywords,
                   php_keyword_num);
    }
  }
  event = inotifytools_next_event(-1);
 }
```

首先是从inotify的消息队列中取出事件，这个函数是个阻塞的函数，因此会一直等到有事件发生。然后进入响应事件的处理流程的循环，接着是判断发生监控事件的文件类型。很抱歉地提一下，目前的版本只支持PHP文件，但要扩展成对JSP的支持也很简单，就是在这里改一下，再添加相应的JSP的webshell和敏感关键字即可。我们团队人手不多，这段时间又在开发一套新产品，所以目前还没添加这个功能。我们做了后续的开发计划，不仅会添加对JSP的支持，还会添加其他功能，逐渐把这个系统完善起来。既然我们把这个程序开源了，也是希望大家能一起来做，毕竟安全这个方面，参与的人越多，考虑得就越全面，安全性就越有保障。

文件类型匹配后就进入到我们自己感觉对这个项目作出了贡献的地方，即webshell和敏感文件判定处理流程。下面展开讨论针对webshell的virusHandler和针对敏感文件的eventHandler这两个函数。事实上，它们的逻辑差不多，在此以virusHandler为例。

```
if (event->mask & IN_CREATE) {
  if (event->mask & IN_ISDIR) {
    inotifytools_watch_recursively(fullPathTemp,
                                   IN_ALL_EVENTS);
  } else {
    tmpkeywords = findViruses(phpVirusfeature,
                              php_virus_num, fullPathTemp);
```

首先判断事件，这里举的是新建事件的例子，从字面IN_CREATE就能看出来。然后是判断触发事件的文件是否是目录，如果是目录就递归监控；如果是普通文件，就进入webshell特征判定流程，如果判定为webshell就入库该事件，否则忽略进入下一个事件处理循环。下面分析一下这个特征判定函数。

为什么说我们认为自己在这个地方做出了一点儿贡献呢？其实Falcon的架构不并复杂，代码的编写也没有用到什么高深的技术，主要就是业务逻辑部分，这是我们在分析了很多webshell之后总结出来的内容，也是Falcon的灵魂所在。以后对Falcon的改进也主要集中在这一部分，把webshell的特征提取得更合理、更精确，最终实现AI。

跟上边一样，webshell和关键字敏感文件是类似的，为了上下文的一致性，我们来看一下webshell的特征来判定。

```
fp = fopen(fileName, "r");

for (i = 0; i < virusesNum; i++) {
  j = 0;
  virustmp =
    (char *) malloc(sizeof(char) * 1024);
  strncpy(virustmp, viruses[i], 1024);
  featureItem = strtok(virustmp, delims);

  while (featureItem != NULL) {
    phpVirusItem[i][j] =
      (char *) malloc(sizeof(char) * 1024);
    strncpy(phpVirusItem[i][j],featureItem,1024);

    if (findKeyWord(phpVirusItem[i][j],
                    fp, &line, source) == 0) {
      sprintf(lineno, "line %d :", line);
      strcat(tmpkeywords, lineno);
      strcat(tmpkeywords, " ");
      strcat(tmpkeywords, source);
      strcat(tmpkeywords, "\r\n");
    }

    j++;
    featureItem = strtok(NULL, delims);
  }
}
```

首先从病毒特征文件中读取特征判定信息，与待

判定文件内容比对。每个病毒的特征是一个字符串数组，依次匹配病毒的每个特征串，完全匹配时就判定为webshell，如果没有匹配任何webshell特征，则认为不是webshell，并进行关键字敏感文件判定。关键字的特征文件跟webshell的特征文件相似。

特征提取

这里举一个简单的例子，提取文本特征串就可以。以下是著名的PHP后门phpspy2011.php自定义的函数encode_pass，是用来加密登录密码的：

```
function encode_pass($pass) {
  $pass = md5('angel'.$pass);
  $pass = md5($pass.'angel');
  $pass = md5('angel'.$pass.'angel');
  return $pass;
}
```

下面是一个功能选项：

```
<a href="javascript:g('secinfo');">Security information</a>
```

提取这两个特征就可以判定是phpspy2011了，格式是：

```
phpspy2011="encode_pass(,secinfo"
```

完整的病毒特征文件在：

https://github.com/secrule/falcon/blob/master/src/conf/phpvirus.conf

Falcon的后续开发计划

目前这个版本的主要目的是抛砖引玉，算是个Demo，只把主要功能展示出来，技术实现上没有仔细的雕琢。程序对外发布后，大家的反馈很让我们感动，因为让我们看到自己的作品是大家都需要的。安全已经从一个奢侈品逐渐变成一个必需品，我们本来就野心勃勃，想把Falcon做成一个简单易用而且功能强大的监控框架——既能监控文件，也能监控系统信息，只要是大家想监控的我们都能监控。使用起来要方便，运行起来要可靠，这是我们的原则。

A计划：开发Unix和Windows版本的Falcon

Unix版本比较容易解决，它有一个和inotify很像的kqueue，基于它编写差不多就是修改函数名。

目前我们还没有对Widows版本很好的想法，这个版本的Falcon和ASP的webshell分析可能要延后。

B计划：基于更新的监控模块Fanotify开发Falcon

Fanotify号称是替代inotify的下一代文件系统通知机制，优点很明显，套用一句奥运口号：“更高，更快，更强”。它更节省系统资源，而且有Access Decision的功能。这个对于杀毒软件很有用。它完成的功能具体说就是，发现病毒文件后不仅可以通知事件发生了，而且能拒绝打开或者执行这个文件，我感觉这个功能很酷。

最终计划：终结者

我们的目的是把这个系统AI化，毕竟webshell是无穷的，我们准备把分析文本特征转为分析行为特征，进而让系统自学习。监控的核心程序我们在用Erlang重写，看看跟C版本的比较一下对于Falcon哪个更合适。现在我们更倾向Erlang，它对于分布式和并发支持很好，很符合Falcon的需要。如果有同样兴趣的朋友，我们希望您能加入进来，让Falcon真正造福于互联网。

我对安全类软件加入开源社区的观点

具体的例子讲完了，务实也要务虚，上面讲了我们在做什么，接下来说说我们之所以要这么做的原因。

开源社区与安全类软件

开源社区一直都很活跃，尤其是这个社区汇集了很多技术高手，软件也是多种多样，其中不乏一些安全类软件，其中有个人防御类、企业级防御类、开发测试类和黑客类等。

在安全界，Backtrack是一个响当当的名字，它是一个基于Ubuntu定制的安全软件平台，现在的版本是Backtrack 5 r2（简称BT5R2），它上面收录的都是开源的安全类软件。下面我就捋着它对安全软件的分类来大致讲一讲。

首先是防御类软件。iptables是Linux上功能全面的防火墙类软件，snort是入侵检测类软件，它们都是开源的。国内现在还有些安全厂商的入侵检测系统是嵌入式的Linux跑一个编写了特定规则的snort，可见其强大。如果我们需要编写一套防

火墙或者入侵检测系统，完全可以参考这两个软件，这就是开源的魅力所在，一切是真正地站在巨人的肩膀上。

接下来是攻击类软件，其实这是把双刃剑，在好人手里，它就是安全检测类的软件，在心怀鬼胎的人的手里，它们就是用来做不法勾当的凶器。

Wireshark是一个开源的网络协议分析工具，也是一个局域网嗅探工具。我们可以用它来学习网络协议的基础知识、测试网络软件的发包情况、网络的运行情况等。因为能抓取到局域网内的网络包，所以明文信息可以轻易获取，嗅探工具嘛，这个就不用多解释了。学习和开发网络底层的同学，这个软件的源码不可不看。

Nmap和Metasploit（简称MSF）是可以放在一起来谈的两个开源工具，前者是个扫描器，后者是个标准化的渗透测试平台。这两个工具可以完美配合，MSF本身内嵌了对Nmap的支持，Nmap的扫描结果，比如网络主机的IP地址，及其对应的开放端口、运行的操作系统和一些应用软件等相应信息入库，在使用MSF进行渗透测试时从数据库中获取这些信息，进行自动化的渗透测试。如果要追求准确率或者要进行深入的渗透测试，则可以不使用这种自动化的方法，选择采用更多人工干预的方式。MSF更强大的地方在于，我们可以添加自己的渗透测试代码，我理解的MSF是开发渗透测试代码的IDE。我们可以把开发好的exploit和payload放到MSF中，方便对软件进行安全测试，MSF现在是基于Ruby开发的，exploit和payload大部分也是用Ruby开发，其代码的可读性高，开发难度小。

提到了exploit的开发就不得不提调试了，下面提到的这个调试工具就不是BT5独有的了，这就是大名鼎鼎的GDB。Windows平台上有很多好用的静态反汇编工具和动态调试工具，这也是得益于Windows的闭源特性，为了做一些底层的驱动开发，不得不做这些工作。Linux是开源的，因此基本不存在Windows面对的问题，BT5提供的逆向工程工具GDB就够用了。它自然也是开放源代码的，有志于驱动开发的朋友值得研究。顺便说一个Windows下很好用的开源调试工具——Immunity Debugger——一个Python开发的工具，也能很好地支持Python插件扩展，值得我们深入学习它的源代码。

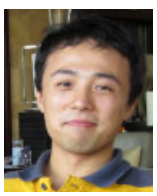
感谢BT5给我提供了这样的进行分类讲述的思路，更感谢它提供了这么多优秀的开源安全软件供我们使用和学习。

安全类软件加入开源社区的意义

开源是趋势，连微软都加入到开源组织OSI了（虽然是老新闻），于是我又开始相信爱情了。在安全越来越受到大家重视的情况下，安全类软件加入到开源社区并成为活跃的一份子就成为了必然。

安全类软件进入开源社区，首先丰富了开源社区，使开源社区更加完善和全面。其次是对安全类软件本身的发展也有很多好处，这样它们就能在开源的生态环境中优胜劣汰，开源社区的人很多都是完美主义的技术人员，他们能甄别出有用的软件并帮助这些经受住考验的软件做得更加完美。当这些安全类的软件发展起来后，工业界必然会受到影响，这有利于信息安全产业的标准化，使信息安全行业的发展更加健康。

作为一名安全研发人员，我深切地体会到开源社区对安全软件的渴望，希望能有更多安全研发人员加入到开源社区，为我们的安全行业做出贡献，为我们的信息环境保驾护航。



李恒磊

Linux C程序员，喜欢鼓捣Python和Erlang，热爱分布式和系统编程，现在专心做安全软件研发。

责任编辑：卢鹃翔（ludx@csdn.net）

多核与异步并行

文 / 陈冠诚

我们在设计多线程程序时往往有很多性能指标，例如低延迟、高吞吐量、高响应度等。随着多核处理器上CPU核数的日益增加，如何高效地利用这些计算资源以满足这些设计目标变得越来越重要。

本文将向读者介绍能够实现低延迟、高吞吐量和高响应度的并行编程技术——异步并行。

在下面的程序中，我们有一个名为do_something()的函数，这个函数实现了将一个文件写入磁盘的功能，执行该函数会占用较多时间。在需要调用该函数时，最简单的用法是对该函数进行同步调用，即下面程序中caller1()所采用的方式。

但这种写法带来的问题是，caller1需要阻塞等待do_something()的完成，期间CPU不能做任何其他计算，从而导致CPU资源的空闲。与此相反，程序中的caller2就采用了异步调用do_something()的方式。caller2在将异步调用do_something的命令发送给worker线程之后，就可以立刻返回并开始执行other_work()，不仅能将other_work()提前完成，更提高了CPU利用率。

```
int do_something(doc)
{
    return write_document(doc); // 耗时的I/O写操作
}

void caller1(doc) {
   result = do_something(doc); //同步调用do_something()
   other_work(); //这个操作需要等待do_something()的完成
   more_other_work();
}
void caller2() {
   worker.send(do_something_msg());//异步调用do_something()
   other_work();
   //这个操作不需要等待do_something()的完成，因此提高了CPU的利用率
   more_other_work();
}
```

在现代计算机体系结构中，I/O设备的速度远比不上CPU，我们做计算的一个基本原则就是在CPU等待I/O请求的同时，分配给CPU足够多的计算任务，从而掩盖I/O请求造成的延迟。在单核时代，我们常使用Multiplexing的方式将I/O任务与计算任务重叠在一起，进而提高程序性能。即一个进程如果进入I/O等待，操作系统会将该进程放入等待队列，并调度执行另一个进程的计算任务。多核时代来临之后，CPU的计算资源变得越来越多，通过使用异步并行技术充分利用CPU的计算资源，提升应用程序的延迟性、吞吐量、响应度也变得越来越普遍。下面通过几个典型应用来对异步并行做更多介绍。

GUI线程的异步并行设计

GUI线程是采用异步并行设计来提高响应度的一个经典例子。一个GUI程序的典型结构是使用一个循环来处理诸如用户点击了某个按钮、系统产生的中断等事件。许多GUI系统还提供了诸如优先级队列等数据结构以保证优先级高的事件能得到及时的响应。下面的代码描述了一个典型的GUI系统。

```
while( message = queue.receive() ) {
  if( it is a "保存文件" request ) {
    save_document(); // 这是一个会产生阻塞的同步调用
  }
  else if( it's a "打印文档" request ) {
    print_document(); // 这是一个会产生阻塞的同步调用
  }
  else
    ...
}
```

该程序存在一个明显的性能Bug：它对save_document()和print_document()这两个非常耗时的操作采用了同步调用的方式，这与GUI线程应该具备及时响应的设计初衷产生了直接矛盾。GUI线程的设计目标不仅是对相应的事件作出正确的响应，更重要的是这些响应必须非常及时。

按照上面这个程序的逻辑，很可能会出现如下情况：用户在点击“保存文件”按钮之后，程序需要花费几秒钟才能完成save_document()的调用，而

程序在这几秒钟内都不能再对其他任何事件作出响应，从而破坏用户体验。

有三种方式可以将耗时的操作从需要保持高响应度的线程中转移出去。下面让我们继续用GUI系统的例子来对这三种方法一一进行介绍，以分析它们各自适用的场景。

方式一：一个专用的工作线程

第一种将耗时操作从GUI线程中转移出去的方式是，使用一个专门的工作线程来异步地处理GUI线程发送的耗时操作请求。如图1所示，GUI线程依次将打印文档（PrintDocument）和保存文档（SaveDocument）两个异步请求发送给工作线程之后就立刻返回，从而继续对用户的其他请求做出及时响应（例如调整窗口大小、编辑文档等）；与此同时，工作线程依次对打印文档和保持文档进行顺序处理，并在该异步请求进行到某一步骤时（或者该异步请求完成时）向GUI线程发送相应的通知信号。

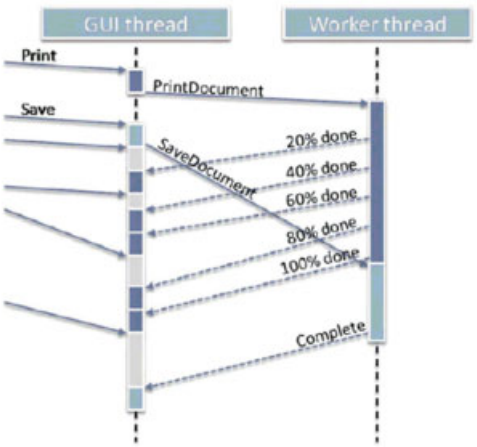


图1　使用专门的工作线程来处理GUI线程的异步请求

让我们来看看这种处理方式的代码：

```
// 第一种方式：使用一个专门的工作线程来处理GUI线程的异步请求GUI线程
while( message = queue.receive() ) {
  if( it`s a "保存文档" request ) {
    worker.send( new save_msg() ); // 发送异步请求
  }
  else if( it`s a "保存文档" completion notification ) {
    display("保存文档成功！"); // 接到异步请求的进度通知
  }
  else if( it`s a "打印文档" request ) {
    worker.send( new print_msg() ); //发送异步请求
  }
  else if( it`s a "打印文档" progress notification ) {
    if( percent < 100 ) // 接到异步请求的进度通知
      display_print_progress( percent );
    else
      display("打印完毕！");
  }
  else
    ...
}
```

```
// 工作线程：处理来自GUI线程的异步请求
while( message = workqueue.receive() ) {
  if( it`s a "保存文档" request )
    save_document(); // 保存文档并在结束后向GUI线程发送通知
  else if( it`s a "打印文档 " request )
    print_document(); // 打印文档并向GUI线程发送进度通知
  else
    ...
}
```

方式二：为每个异步请求分配一个工作线程

在第一种方法的基础之上，我们可以做一些相应的扩展：对每个GUI线程的异步请求都分配一个专门的工作线程，而不是只用一个工作线程去处理所有异步请求。这种方式的好处很明显，异步请求被多个线程分别并行处理，因此提升了处理速度。值得注意的是，我们需要及时对这些工作线程进行垃圾回收操作，否则大量线程会造成内存资源的紧张，如图2所示。

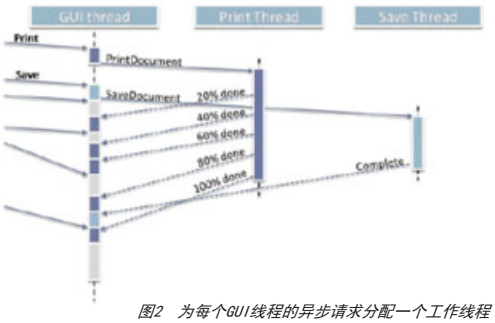


图2　为每个GUI线程的异步请求分配一个工作线程

这种模式的代码如下所示。因为针对每个异步请求都启动一个新的线程，所以我们可以充分地利用多核的计算资源，更快地完成相应的任务。

```
// 方式二：每一个异步请求分配一个线程
while( message = queue.receive() ) {
  if( it`s a "保存文档" request ) {
    ... new Thread( [] { save_dcument(); } );
    // 启动新线程对异步请求进行处理
  }
  else if( it`s a "打印文档" request ) {
    ... new Thread( [] { print_document(); } );
    // 启动新线程对异步请求进行处理
    }
  else if( it`s a "保存文档" notification ) { ... }
  // 同方式一
  else if( it`s a "打印文档" progress notification ) { ... }
  // 同方式一
  else
    ...
}
```

方式三：使用线程池来处理异步请求

第三种方式更进一步：我们可以根据多核硬件资源的多少来启动一个专门的线程池，用线程池来完成GUI线程的异步请求。这种方式的好处在于，我们可以在充分利用多核硬件资源，以及并行地对异步请求进行高效处理间取得一个很好的平衡。该方式的工作示意如图3所示。

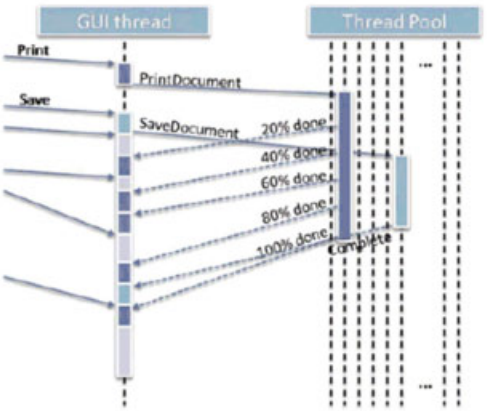


图3　使用线程池来处理GUI线程的异步请求

让我们来看一下这种方式的伪代码。需要注意的是，线程池的具体实现每种语言各有不同。

```
// 方式三：使用线程池来处理异步请求
while( message = queue.receive() ) {
  if( it`s a "保存文档" request ) {
    pool.run( [] { save_document(); } ); // 线程池的异步调用
  }
  else if( it`s a "打印文档" request ) {
    pool.run( [] { print_document(); } ); //线程池的异步调用
  }
  else if( it`s a "保存文档" notification ) { ... }
  // 同前
  else if( it`s a "打印文档" progress notification ) { ... }
  // 同前
  else
    ...
}
```

Grand Central Dispatch的异步并行

Grand Central Dispatch（GCD）是苹果于Mac OS X 10.6和iOS4中发布的一项并行编程技术。对使用GCD的程序员来说，只需要将被处理的任务块丢到一个全局任务队列中，这个任务队列会由操作系统自动地分配和调度多个线程来进行并行处理。将需要被处理的任务块插入到任务队列中的方式有两种——同步插入和异步插入。

让我们看一个使用GCD异步并行的实例。在下面的程序中，analyzeDocument函数需要完成的功能是对这个文档的字数和段落数进行相关统计。在分析一个很小的文档时，该函数可能很快就执行完毕，因此在主线程中同步调用这个函数也不会有很大的性能问题。但如果文件非常大，该函数可能变得非常耗时，而如果仍然在主线程中同步调用该方法，就可能带来很大的性能延迟，从而影响用户体验。

```
// 不使用GCD的版本
- (IBAction)analyzeDocument:(NSButton *)sender {
  NSDictionary *stats = [myDoc analyze];
  [myModel setDict:stats];
  [myStatsView setNeedsDisplay:YES];
  [stats release];
}
```

使用GCD的异步并行机制来优化这个函数非常简单。如下所示，我们只需要在原来的代码基础上，先通过dispatch_get_global_queue来获取全局队列的引用，然后再将任务块通过dispatch_async方法插入该队列即可。任务块的执行会交由操作系统去处理，并在该任务块完成时通知主线程。一般来讲，异步插入的方式拥有更高的性能，因为在插入任务之后dispatch_async可以直接返回，不需要进行额外等待。

```
//使用GCD异步并行的版本
- (IBAction)analyzeDocument:(NSButton *)sender
{
  dispatch_queue_t queue =
    dispatch_get_global_queue(DISPATCH_QUEUE_
                              PRIORITY_HIGH, 0ul);
  dispatch_async(queue, ^{
      NSDictionary *stats = [myDoc analyze];
      [myModel setDict:stats];
      [myStatsView setNeedsDisplay:YES];
      [stats release];
    });
}
```

总结

本文对多核编程时常用的异步并行技术做了相关介绍。通过使用异步并行技术，我们可以将耗时的操作交给别的线程处理，主线程因此可以处理其他更有意义的计算任务，从而有效改善系统的延迟性、吞吐率和响应性。

陈冠诚

IBM中国研究院研究员，主要研究方向为并行计算、计算机体系结构及大规模分布式系统。个人博客为http://parallellabs.com。

责任编辑：卢鹃翔（ludx@csdn.net）

名人堂

Google帝国背后的英雄

Urs Hölzle

文 / 陈粲然

Urs Hölzle是Google的基础架构高级副总裁。作为Google的前10个工程师，他为Google的基础架构建设贡献颇多。

1988年，Urs Hölzle在家乡的苏黎世联邦理工学院获得计算机科学博士学位并获得了富布赖特科学奖学金。1994年，他又在斯坦福大学获得博士学位，那时他的研究内容集中在编程语言及其高效实现。他和David Griswold（现Google通信研究员）以及Lars Bak（Google Chrome V8的开发者）一道基于这项研究开发了一个名为HotSpot的高性能Java虚拟机，被Sun公司的业务部门JavaSoft（后更名为Sun Microsystems)于1997年收购，作为Sun最初的Java虚拟机。在加入Google之前的日子里，他在加州大学圣塔芭芭拉分校计算机科学系担任副教授。

他设计并领导Google建立了极其高效的数据中心，据说这个数据中心所耗费的能源还不到传统数据中心的一半。

从存储角度来看，Hölzle的策划内容主要是指Google File System（GFS），这个系统包括数千台基于Linux系统的普通服务器。这些服务器被集群在一起，处理大型共享文件，这些文件大小通常可到达几GB。Google搜索试验室网站上的一份记载显示，该公司最大的集群通过1000多台机器上几千块磁盘，提供了数百TB的存储容量。

Google目前使用的模式是：非定制与定制软件相结合，并在普通电脑上运用这些软件。该架构简化了大量数据的存储与处理过程，也使得Google能够很便捷地提供面向全球的大型产品与服务。同时，也实现了对大范围计算机集群的管理自动化。

2007年，他带领Google和Intel一起推行“气候救助者计算公开行动”（Climate Savers Computing Initiative），这项环保节能计划旨在使个人电脑和服务器能够节约更多的能量和减少二氧化碳等温室有害气体的排放量。他们力图建立更低的计算机能耗标准和更有效的能源管理软件，该计划要求计算机的有效能源利用率达到90%，减少在计算过程中发热消耗的电力，尤其是在大的服务器中心用于空调的电力。这项新计划的签约者还包括戴尔、惠普、IBM、联想、微软、美国环境保护署以及超过25个的环境保护团体、有志于能源节约的公司和大学。

而在2011年，Hölzle宣布Google的慈善机构Google.org在替代能源投资战略上的转变：他们决定抛弃发展太阳能发电技术，因为它的费用难以与价格不断下降的太阳能光电技术相匹敌。他表示Google已经立志在2012年时创建一个可供50000个美国家庭使用的高达50兆瓦特的设备，并且该设备具有可恢复的发电能力。

目前，在Hölzle的领导下，Google已利用OpenFlow技术整改了Google各大数据中心的内部网络中的很大一部分，以帮助公司提高效率。据Hölzle称，这次整改背后的想法是Google史上在网络方面最重大的变革。OpenFlow系统可以统观全网的状态，测算数据传输所需时间，并自动提供数据传输的最佳路由。而且，OpenFlow还会自动分配数据传输的优先级。这样，对网络管理者而言，他们的管理成本可以大幅降低。

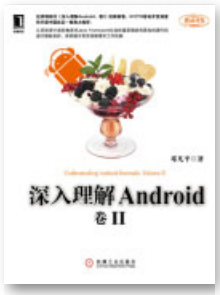
工作之余，他和Luiz Barroso（Google著名工程师）一同撰写了*The Datacenter as a Computer*和*The Case for Energy Proportional Computing*两本书，向业界分享Google的成功经验。他们在The Datacenter as a Computer中介绍了数据中心的设计方案，这本书被出版社放在网上供大家免费下载阅读。而在*The Case for Energy Proportional Computing*中，他们提出了“服务器在设计时应该让其工作电流和负荷电流成比例，因为它们在运行时并不总是处于满载状态”这一观点。

虽然在工作中从事着Google的技术核心工作，但他行事一直非常低调，除了在2005年的愚人节推出了饮料Google Gulp外，没有太多轶事可寻。但就像世人知道的那样，Google能建立如今的帝国，离不开Urs Hölzle的汗马功劳。

推荐图书

《深入理解Android》背后的故事

文 / 邓凡平



深入理解Android：卷II

作者：邓凡平

我工作的大部分内容是维护已有Android系统，也就是修改已有系统的Bug，因此，经常会将对Android的理解和Bug关联到一起去。就Audio而言，我能很快说出相关的Bug以及修改情况，却不能说出整个Audio的工作原理。这种只见树木不见森林的情况，令我对自己非常不满，因此，下定决心要尽快打通各知识点，搭建一个较为完整的Android知识框架。于是，我开始了Android研究之旅。

Android中各个模块的研究方法是我一直在琢磨的事情。对于Native Framework层来说，其主要关注数据流程，例如Audio、Surface系统等，因此，用Source Insight多跟踪几遍代码就能弄明白。但对于Java Framework来说，其关注点是管理和交互规则。例如ActivityManagerService内部有大段代码来处理Activity不同启动模式（如SingleTop、SingleInstance等）的情况。我之前也采用Source Insight加“蛮看”的方式。结果整整2周，毫无进展，因为分支太多，堆栈太深，很难理解清楚。

怎么办呢？能亲自动手调试相关模块当然最好。于是，我花了一整天的时间从网上找调试System_Process的方法，几乎把Google的一个开发者论坛翻了个底朝天，才找到一篇帖子。它指向的却是Android网页上的一个链接http://source.android.com/source/using-eclipse.html。

掌握调试方法后，我又开始犯难。那时Android 4.0刚出来，一定要用真机调试吗？几经周折，我才想到模拟器加Eclipse这对黄金搭档，它们极大地提高了研究速度。但整个ActivityManagerService研究下来也花了将近1个月的时间，但收获很大。我及时将它们整理成文章，最终成为了撰写《深入理解Android》系列书的一手素材。

结合我的工作经验和研究Android过程中的收获，我有以下两点建议。

要从“基于Android并高于Android”的角度来看待和分析Android。对于卷I的读者来说，他们可能是芯片厂商、底层厂商的工程师居多。他们以后转向其他平台的可能性非常大。因此，一定要站在更高的角度来学习Android，而不应该局限在这个平台。

要初步具有大型复杂代码的分析能力。在卷II的读者中Java程序员比较多。Java有很多优秀的开源框架，但大部分程序员不会去研究其中的细节。对他们来说，目前缺乏的是大型复杂代码的分析能力（当然，如果你能完整看完《Linux内核情景分析》一书，则不属于此范畴）。这个能力实际上是内功，需要踏踏实实，潜心钻下去才能修炼得到。

本月新书

HTTP权威指南

作者：David Gourley Brian Totty Marjorie Sayer Sailu Reddy Anshu Aggarwal

译者：陈涓，赵振平

本书不仅仅是一本HTTP首部参考手册，还是一本名副其实的Web架构“圣经”。它深入说明了Web的工作原理，梳理了HTTP中一些互相关联且常被误解的规则，并详细介绍了HTTP各方面的特性，尤其对HTTP“为什么”这样做进行了详细的解释，而不仅仅停留在它是“怎样做”的。一个假想的在线五金与家装商店示例“Joe的五金商店”贯穿全书，非常有助于理解书中的技术概念。作者还特意为这个商店构建了一个真实的Web站点，以便大家能够测试书中的部分实例。

深入理解Oracle Exadata

作者：Kerry Osborne Randy Johnson Tanel Poder

译者：黄凯耀，张乐奕，张瑞

Oracle Exadata Database Machine，简称Exadata，是2008年9月Oracle在甲骨文全球用户大会上推出的集成一体化数据库机。刚一推出便立即引起了业界人士的关注，并很快得到了全球用户的广泛认可。随着Exadata产品和技术的不断更新和广泛应用，很多Exadata技术爱好者都希望有一本深入介绍Exadata的技术图书。本书应运而生，它清晰诠释了Exadata，详细说明了该系统是如何把服务器、存储和数据库软件技术结合在一起，以成为一个适合事物处理和数据仓库应用的系统。

为什么要学习多种编程语言

文 / 卢鹆翔

七周七语言

作者：Bruce A.Tate
译者：巨成，戴玮，白明

Google研发总监Peter Norvig曾谈到他在编程上成功的秘诀之一：“至少要学会六门编程语言。一种面向对象语言，如Java或C++；一种函数式语言，如Lisp或ML；一种支持语法抽象的语言， 如Lisp；一种声明式语言，如Prolog；一种支持协同式编程的语言，如Icon或Scheme；还有一种支持并行的语言，如Sisal。”

《UNIX编程艺术》作者Eric Raymond也表达过相似观点：“Python、C、Perl和Lisp除了是最重要的四种基本语言，它们还代表了四种非常不同的编程方法，每种都会让你受益匪浅。”

编程语言会影响你思考问题的方式，因为每种语言所包含的一系列特性——静态类型和动态类型、早期绑定与延迟绑定——都会鼓励你采用某种特定的解决方案。Raymond甚至认为，即使你并不真的使用Lisp，单是学习它就会让你成为更好的程序员——假如你只使用命令式语言，但只要头脑中思考着函数式方案，写出的代码也许就会不同。John Carmack在《用C++进行函数式编程》（http://www.programmer.com.cn/12717/）中就谈到过他的体会。

维特根斯坦曾说“语言的界限就是我与世界的界限”，新学会一门编程语言，才有可能领略由这门语言搭建起来的软件世界的精妙。松本行弘曾谈到，他在设计Ruby时，Emacs带给他极大的灵感，假如他不曾深入研究Emacs Lisp，我们看到的Ruby不会是今天的样子。

《七周七语言》中介绍了Ruby、Io、Prolog、Scala、Erlang、Clojure、Haskell，与Norvig给出语言列表覆盖的范围相近，而且语言更“现代”，是一本了解编程范型理想的入门书。

但光单看菜谱成不了好厨师，真正学好编程是个复杂的活儿，只靠书本和课程都做不到。几乎所有的编程好手都是自学成材，真正能起作用的就是去亲自读代码、写程序。

如今寻找代码已经变得很容易，无论你对其中哪种语言好奇，总能很方便地在GitHub上按照分类找到有趣的应用。

读完这本书，如果你对编程语言这个话题本身感兴趣，还可以阅读探讨编程语言特性的进阶书，比如《程序设计语言：实践之路》。此外，一些语言中极有用但更高阶的技巧，在这本书中并没有探讨。假如你想见识Clojure中宏的威力，可以阅读Paul Graham的《On Lisp》，网络上还可以找到将这本书案例用Clojure改写的版本。

《程序员修炼之道》的作者David Thomas鼓励程序员每年学习一门新语言，因此，在读完这本书之后，请不要停下你继续学习其他编程语言的脚步。

Linux命令行与Shell脚本编程大全（第2版）

作者：Richard Blum　Christine Bresnahan

译者：武海峰

本书从“初识Linux shell”讲起，描述了构成整个Linux系统的各个部分，并且说明了Shell是如何融入Linux的。随后，由浅入深讲述了Shell方方面面的内容，同时涵盖了详尽的动手教程和实际应用中的实用信息，提供了相关参考信息和背景资料。“如果你想从整体上了解Linux并开始学写脚本，那么就从本书开始吧。”亚马逊上的这条读者评论，道破了本书最核心的价值——内容全面、条理清晰，非常有助于读者系统性地学习和研究Linux命令行和Shell脚本编程。

HTML5游戏开发实战

作者：Makzan

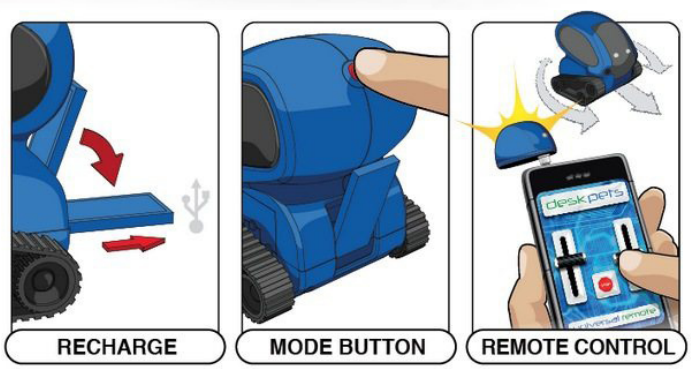
译者：吕定平，陈升想

目前，HTML5、CSS3以及相关的JavaScript API是一个热点话题，因为它们带来一个新的游戏市场——HTML5游戏。本书清晰而全面地展示了如何使用最新的HTML5和CSS3标准来构建各种常见类型的游戏，围绕纸牌游戏、绘图游戏、物理游戏和多人游戏精心组织了6个富有趣味性和技术性的游戏案例。动手实践这些案例，不仅能掌握各种类型游戏的开发思路和设计方法，而且还能掌握HTML5和CSS3技术中与游戏开发相关的理论知识。

GEEK产品

TankBot

美国Desk Pets出品的TankBot，支持Android系统和iOS系统的手机，支持USB充电。除了手机App操控之外，它还具备“自动导航无人驾驶功能”，平时放在桌面让它自动遛弯，其自带的红外线传感器可绕过障碍。迷宫模式、自由漫游模式以及遥控模式。右侧为TankBot操作示意图。



Fish Eye Lens

iPhone增强式镜头的配件Fish Eye Lens鱼眼镜头的整体尺寸为12.5 x 7 x 2.5厘米，重量仅45克，由橡胶材质制的背壳和可拆卸的鱼眼镜头组成。镜头的广角度为180°，可放大0.33x，物镜直径为23mm。极端且变形、透视感强的鱼眼镜头能为你带来全新的视觉冲击，适用于那些富有想象力和勇于挑战的摄影者。

XDJ-Aero

日本先锋DJ近日推出了一款Wi-Fi DJ控制器。你可以用设备下载好相对应的应用程序后，与XDJ-AERO进行连接，然后利用它来进行音乐的控制和管理。XDJ-AERO可连接最多四台终端（iOS/Andriod/MAC/PC），还有一个24位音频口和MIDI控制器。

Pet's View

概念相机Pet's View是一款宠物专用照相机，它是由一个带USB插口的相机和可调节松紧的脖带组成。在相机内部有一个传感器，可以监测宠物的心跳和体温，当宠物兴奋时就会触发相机进行拍摄。相机的电力则由装在脖带上的太阳能电板提供。

The Moshi Matic Washing Machine

这款全新的洗衣机采用了最新的AMR机器人技术，人们可进行远程操作，通过手机传送一系列的命令，来设定洗衣时间，查看清洗程度和模式。

Polaroid Z2300

宝丽来推出数码版即显Z2300，配置如下：1000万像素、6x光学变焦、3寸屏幕、720p摄像、SD卡扩充（机身自带32MB）、5种闪光模式，即时打印功能配合Zink技术可以打印出2x3寸照片。机身售价为169美元。

0

```
// 写这段代码的时候
// 只有上帝和我知道它是干嘛的
// 现在，只有上帝知道
```

1

```
// 虽然我也不想这样做，
// 但是有个讨厌的家伙就要我这样做，
// 迟早会证明，他是错的。
// 2011-12-21
```

2

```
/* 如果你能弄清楚这一段，求
求你务必发邮件至aaa@bbb.com
告诉我，不胜感激 */
```

3

```
/*勇士，乐观地接收这个现实吧！
2011-11-23*/
```

4

```
// 这个不知道是谁写的
// 看起来没用
// 但我不敢删
```

5

```
// 去他妈的。就这样发布吧。
```

6

```
/* 魔法。勿动！ */
```

7

```
/* 致终于来到这里的勇敢的人：
你是被上帝选中的人，英勇的、不辞劳苦
的、不眠不休地来修改我们这最棘手的代
码的编程骑士。你，我们的救世主、人中
之龙，我要对你说：永远不要放弃，永远
不要对自己失望，永远不要逃走。永远不
要哭泣，永远不要说再见。
永远不要说谎来
伤害自己。*/
```

8

```
// 亲爱的维护者：
// 如果你尝试了对这段程序进行“优化”，
// 并认识到这种企图是大错特错，
// 请增加下面这个计数器的个数，
// 用于对后来人进行警告：
// 人们总共在这里浪费了 = 39h
```

原载于stackoverflow
翻译改编自《外刊IT评论》

西乔
设计师，项目经理。
2006年起携创业团队从事Web
技术外包开发及产品咨询顾问。

如果你有什么好玩的关于程序员的故事、对话、代码，
愿意通过漫画的形式分享，
请给西乔发邮件：arthur369@gmail.com